Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Trabajo Fin de Máster
Máster en Ingeniería de Sistemas e Informática

Characterization of Interconnection Networks in CMPs Using Full-System Simulation

Marta Ortín Obón

Directores: María Villarroya Gaudó y Darío Suárez Gracia

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2011/2012
Septiembre 2012

Acknowledgements

First of all, I would like to thank my advisors, María and Darío, for all their help and guidance.

I would like to thank Cruz for all her help with interconnection networks and her fast replies to every email aswering all of our questions. I look forward to continuing working with you.

Special thanks to Jorge, who has been helping me every day with all the technical details and problems that came up with the simulations.

I also thank all the members of the Computer Architecture Group (gaZ), especially Víctor, for all his advice and useful insights, and Chus, who made sure the cluster was fixed as soon as possible in the middle of the summer.

Caracterización de redes de interconexión en CMPs mediante simulación de sistema completo

Resumen ejecutivo

Los computadores más recientes incluyen complejos chips compuestos de varios procesadores y una cantidad significativa de memoria cache. La tendencia actual consiste en conectar varios nodos, cada uno de ellos con un procesador y uno o más niveles de cache privada y/o compartida, utilizando una red de interconexión. La importancia de esta red está aumentando a medida que crece el número de nodos que se integran en un chip, ya que pueden aparecer cuellos de botella en la comunicación que reduzcan las prestaciones. Además, la red contribuye en gran medida al consumo de energía y área del chip.

Actualmente, hay numerosos estudios centrados en las redes de interconexión, pero la mayoría no modelan en detalle todos los componentes del sistema debido a la alta complejidad que esto conlleva. Además, suelen utilizar tráfico sintético o trazas de aplicaciones que no son representativas del comportamiento real de un programa.

En este proyecto, comparamos el comportamiento de tres topologías: el anillo bidireccional, la malla y el toro. El anillo es una topología mínima con bajo coste en energía pero peor rendimiento debido a la mayor latencia de comunicación entre nodos. Por otro lado, el toro tiene mayor número de enlaces entre nodos y ofrece mejores prestaciones. La malla ha sido incluida como una opción intermedia altamente popular. Analizaremos también dos topologías de anillo adicionales que aprovechan la reducida área y complejidad del mismo: una con mayor ancho de banda y otra con *routers* de menor número de ciclos.

Modelamos cuidadosamente todos los componentes del sistema (procesadores, jerarquía de memoria y red de interconexión) utilizando simulación de sistema completo. Ejecutamos aplicaciones reales en arquitecturas con 16 y 64 nodos, incluyendo tanto cargas paralelas como multiprogramadas (ejecución de varias aplicaciones independientes). Fue llevado a cabo un estudio del comportamiento de una de las *suites* de cargas paralelas incluidas en este proyecto y se presentaron los resultados en una sesión de pósters en una conferencia internacional.

Demostramos que la topología de la red afecta en gran medida al rendimiento en sistemas con 64 nodos. Con las topologías de anillo, los tiempos de ejecución son mucho mayores debido al aumento del número de saltos que le cuesta a un mensaje atravesar la red. El toro es la topología que ofrece mejor rendimiento, pero la elección más óptima sería la malla si tenemos en cuenta también energía y área. Por otro lado, para chips con 16 nodos, las diferencias en rendimiento son menores y un anillo con routers de 3 cyclos ofrece un tiempo de ejecución aceptable con el menor coste en área y energía.

Nuestra aportación más significativa está relacionada con la distribución del tráfico en la red. Vemos que el tráfico no está distribuido uniformemente y que los nodos con mayores tasas de inyección varían con la aplicación. Hasta donde nosotros sabemos, no hay ningún trabajo de investigación previo que destaque este comportamiento.

Como trabajo futuro, propondremos un diseño para la red que ofrezca al mismo tiempo buen rendimiento y bajo consumo energético, partiendo de las conclusiones obtenidas de este estudio.

Characterization of Interconnection Networks in CMPs Using Full-System Simulation

Abstract

In modern computer architecture systems, chips are composed of several processors and a significant amount of memory. The current trend involves the interconnection of several nodes, each of them with a processor and one or more levels of shared and/or private memory caches. The interconnection network is responsible for connecting the nodes. The importance of the interconnect is growing as the number of nodes integrated in a chip increases, because communication among nodes can become a bottleneck and compromise performance improvement. It also contributes with a substantial share to power consumption and chip area.

There are many studies that focus on the interconnection network, but most of them do not model in detail all the components of the system due to the complexities it involves. Besides, they usually only simulate synthetic traffic or application traces that are not representative of the behaviour of real applications. They also center their conclusions on the interconnect, failing to analyse the impact on overall performance and the effects some components of the chip may have on others.

In this work, we compare the behaviour of three topologies: bidirectional ring, mesh and torus. The ring is a minimal topology that has lower energy costs but worse performance, that is, higher average communication latency between nodes. On the other hand, a torus has more links between nodes and offers the best performance. The mesh topology has been included as a very popular intermediate option. We include two additional ring topologies that benefit from the reduced area and complexity: one with increased bandwidth and another with reduced-pipeline routers.

We carefully model all the components of the system (processors, memory hierarchy and interconnect) using full-system simulation. We execute real applications on a 16 and a 64-core system, including both parallel (execution of a multithreaded application) and multiprogrammed workloads (execution of several independent applications). We performed a complete study of one of the benchmark suites featured in this project and presented our results in a poster session at an international conference. We analyse network centered metrics, like latency, injection rate, hop count and congestion, as well as chip-level metrics, such as performance, area and power.

We prove that performance is highly affected by the choice of the interconnect in 64-core systems. The ring topologies produce much larger execution times due the increased number of hops it takes to traverse the network. The torus has the best performance, but the mesh would be the best choice if we also consider area and power. On the other hand, for 16-core chips, differences in performance are not so big and a ring topology with 3-cycle routers offers acceptable execution time with the lowest power consumption and area requirements.

Our main contribution is related to the distribution of traffic on the network. We show that traffic is not uniformly distributed on the network and that the most highly used areas of the chip depend on the application. As far as we know, there is no previous research where this behaviour has been noted.

In our future work, we will propose a network design that offers good performance as well as low energy consumption, incorporating the conclusions drawn from this study.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Interconnection networks, both system-level and chip-level, have been an important part of computer architecture since the beginning of computing in the sixties. Today, interconnection networks are used in supercomputers, data centers, for input/output and to communicate processors on a chip. In this work, we are focusing on networks on chip (NOCs).

Nowadays, chips are composed of several processors and a significant amount of memory. A popular trend in the organization of general purpose chips consist on interconnecting several nodes (usually called tiles), each of them with a processor (core) and one or more levels of shared and/or private memory caches. Nodes communicate through an interconnection network that allows them to exchange information (both pair to pair and multicast and broadcast messages). In these general-purpose chips we can execute independent applications on each node (so as to increase throughput) or parallel applications (in order to reduce execution time).

If the current trend continues, the scale of integration, along with the emergent 3D stacking technology, will allow us to exponentially increase the number of nodes in a chip, reaching hundreds or even thousands of cores in less than ten years. So that this technology evolution can be translated into an improvement on performance, the capacity of communication among tiles should scale in the same proportion.

At these time, there are very few studies that model in detail the set of processors, interconnection network and memory hierarchy, due to the complexities it involves. Besides, analysis that focus on interconnection networks are usually performed simulating synthetic traffic or application traces that do not capture the behaviour of a real execution. They also center their conclusions on the interconnect, failing to analyse the impact on overall performance.

In order to get representative results, we analyse the behaviour of real applications on the network, carefully modeling all the components mentioned earlier. We include both parallel and mutiprogrammed workloads. This allows us to study the effect of the interconnection network configuration on the whole system and the interactions between the memory subsystem and the interconnect. Specifically, we compare the performance of three topologies (bidirectional ring, mesh and torus), simulating a chip multi-processor (CMP) with 16 and 64 cores. The ring is a minimal topology that has lower energy costs but worse performance, that is, higher average communication latency between nodes. On the other hand, a torus has more links between nodes and offers the best performance. The mesh topology has been included as a very popular intermediate option. We include two additional ring topologies that benefit from the reduced area and complexity: one with increased bandwidth and another with reduced-pipeline routers.

We wanted to determine whether a higher energy consumption is justified or not and if it is necessary to improve the efficiency of the network or if, on the contrary, it is over dimensioned and we should introduce power reduction policies.

Based on our conclusions, we plan to continue our work with a PhD thesis in which we will propose a network design that will offer good performance as well as low energy consumption. We will try to reduce the latency of the network and use our knowledge of the requirements imposed on the interconnect by the memory subsystem.

1.1. Project Development

This master's thesis has been developed in the Computer Architecture Group at the University of Zaragoza (gaZ) and has been supported by grant TIN2010-21291-C02-01 (Spanish Government and European ERDF). It started in February 2012 and was completed in September 2012 with full-time dedication.

At the beginning of the master's thesis I held a grant from the *Instituto Universitario de Investigación e Ingeniería de Aragón(i3A)* and then, I got a four-year research grant from the *Gobierno de Aragón*. During the course of the project I attended several conferences and seminars, including the NaNoC summer school on networks on chip. This is a summer school hosted by the NaNoC project (FP7 ICT), which aims to develop a network on chip design platform. It took place in Munich from the 11^{th} to the 13^{th} of June 2012. I also presented an analysis of one of the benchmark suites included in this project in the HiPEAC conference, which took place in Paris from the 23^{rd} to the 25^{th} of January 2012.

We have written a paper that sumarizes this work and will be submitted to the *Interconnection Workshop on Network Architectures: On-Chip, Multi-Chip*. The workshop will be co-located with the HIPEAC conference that will take place in 2013 in Berlin.

1.2. Goals of the Master's Thesis

The aim of this project is to perform a comprehensive analysis of the behaviour of real applications on the interconnection network in chip multiprocessors (CMPs). In particular, it includes the following tasks:

1. Study of the state of the art on interconnection networks: architecture, organization and implementation.
2. Analysis of the GARNET network simulator [1], which is included in the GEMS module [33] and used in combination with Simics, a full-system simulator [31]. We need to model the topologies and establish the configuration parameters. It is also necessary to adapt the implementation of the routers to our necessities and include the calculation of new statistics.
3. Selection of the simulation workload. We use parallel programs from the PARSEC and SPLASH-2 benchmark suites and build multiprogrammed workloads from SPEC CPU2006 applications.
4. Measurement of energy and area costs of the topologies and comparison with those of the memory hierarchy using high-level circuit-modeling tools, such as CACTI [35] and Orion [23, 24].

5. Evaluation of the impact of different network configurations on the overall performance of the system and drawing of conclusions about the tradeoffs on performance and power consumption.

With the completion of the former tasks described in this report, all the goals of this master's thesis have been met.

1.3. Organization of the Report

The rest of this document is organized as follows: Chapter 2 presents the state of the art on interconnection network research; Chapter 3 describes the main characteristics of the CMP we are modeling, with special attention to the interconnection network; Chapter 4 explains the methodology followed in our experiments; Chapter 5 summarizes the results of the study and Chapter 6 concludes the report.

The document includes the following appendices:

A. Project management. It contains information about the time invested in each section of the project and some problems we have had to deal with.

B. CMP Architecture and Memory Subsystem. It includes a detailed description of our system, explaining the general layout and the memory coherence protocol.

C. Methodology and experimental environment. It focuses on the metrics used to analyse the impact of the interconnect on the system, the workload used and the tuning of the simulator.

D. Results. It includes a complete analysis of our results.

E. Research paper. It includes a paper that sumarizes this work and will be submitted to an upcoming international conference.

Chapter 2
State of the Art

In this chapter we summarize previous work that analyses and tries to improve the performance of interconnection networks, focusing on networks on chip.

There have recently been numerous papers that focus on networks on chip, since the interconnect has been identified as a critical part of the system that has a great influence on overall performance, energy consumption and chip area requirements. They focus on tiled multiprocessors with shared memory.

There are many papers that propose alternatives to the most commonly used router architectures, topologies and flow control methods, but none of them model the impact of their contributions by running real programs on a full system. Among that research, we highlight the following: Carara *et al.* propose to revisit circuit-switching which, as opposed to packet-switching, allows to reduce buffer size and guarantees throughput and latency [12]; Walter *et al.* try to avoid hotspots on systems on chip by implementing a distributed access regulation technique that fairly allocates resources for those modules [44]; Mishra *et al.* propose an heterogeneous on-chip interconnect that allocates more resources for routers suffering higher traffic but they only get good results with a mesh topology [34]; Koibuchi *et al.* detect that adding random links to a ring topology results in big performance gains, although they only experiment with a network simulator [25]. All these studies either do not model the whole system, do not include a significant variety of real workloads or do not experiment with different topologies. Also, most of them only include network-related metrics and fail to report on overall performance, or elaborate conclusion based on IPC (instructions per cycle), which has been reported to be unsuitable for CMPs [50].

It is worth mentioning another approach to network on chip research. Instead of dealing with classical network issues, there is previous work that tries to improve performance based on the known behaviour of the memory subsystem and the coherence protocol. A selection of recent papers follows. Yoon *et al.* propose an architecture with parallel physical networks with narrower links and smaller routers that eliminates virtual channels [48]. Seiculescu *et al.* propose to use two dedicated networks, one for requests and one for replies [39]. Lodde *et al.* introduce a smaller network for invalidation messages, but only tests their design with memory access traces [28]. Agarwal *et al.* propose embedding small in-network coherence filters inside on-chip routers to dynamically track sharing patterns and eliminate broadcast messages [2]. These studies try to improve the performance of the most commonly used networks, but do not venture with less conventional topologies. Also, they only experiment with a maximum of 16 cores. Krishna *et al.* propose a system to improve the frequent 1-to-many and

many-to-1 communication patterns by forking and aggregating packets to avoid the increment in the amount of traffic when scaling the number of nodes [26]. Bezerra *et al.* try to reduce traffic by statically mapping memory blocks to physical locations on the chip that are close to cores that access them [6]. The last two proposals are only evaluated with a typical mesh topology.

There are very few papers which focus on the comparison of interconnection network configurations. Balfour and Dally present an analysis of how different topologies affect performance, area and energy efficiency [3]. However, they do not model the memory subsystem, they only use synthetic traffic patterns and they fail to include a simpler topology like the ring. Sanchez *et al.* explore the implications of interconnection network design for CMPs, but they focus on more complex topologies, such as the fat three and the flattened butterfly, and they do not include multiprogrammed workloads [37]. There is no clear information about the way they perform their simulations. Therefore, we are not sure about how accurately they are reproducing real system behaviour. Their results point out that the main parameter that influences performance is the number of hops the messages will need to get from one tile to another in the network. They also highlight the need of a careful codesign of the interconnection network and the cache hierarchy. This necessity has also been noted by Kumar *et al.*, but their research does not go above 16 cores [27].

There are also proposals that try to reduce the time design-space exploration takes. Hestness *et al.* propose a new trace-based network simulation methodology that does not ignore dependencies between messages. Simulation with this method is faster than full-system simulation while still having high fidelity [19]. It allows us to experiment with several interconnect configurations with little overhead. The drawback is that the traces have to be rebuilt if we want to test different number of cores or change parameters in the memory hierarchy. Knowing that the codesign of all the elements of the system is critical for a balanced CMP, most benefits from this method are overridden.

Table 2.1 sums up the characteristics of the papers mentioned above and allows us to find the most common simulation techniques and configurations at a glance. The last column includes the characteristics of this work, which combines features that others do not integrate.

In this work we present an analysis of three topologies with varying degrees of complexity, performance and power and area costs. We perform full-system simulation of real workloads and carefully model both the memory subsystem and the interconnect. Our aim is to extract meaningful conclusions that will indicate the weaknesses of current configurations and guide our future research.

Table 2.1: Characteristics of the research included in recent papers about interconnection networks. The first rows in *Simulation methodology* refer to the simulation of the interconnect without including other elements of the system like the processors or memory subsystem.

	References														
	[12]	[44]	[34]	[25]	[48]	[39]	[28]	[2]	[26]	[6]	[3]	[37]	[27]	[19]	This work
Simulation methodology															
Interconnect with synthetic traffic	X	X	X	X	X						X				
Interconnect with traces							X								

	References														
	[12]	[44]	[34]	[25]	[48]	[39]	[28]	[2]	[26]	[6]	[3]	[37]	[27]	[19]	This work
Interconnect with enhanced traces													X	X	
Interconnect with one real app.		X													
Full-simulation with real apps.			X		X	X		X	X	X		X			X
Include multiprogr.						X									X
Topologies															
Bus, crossbar													X		
Ring				X											X
Mesh	X	X	X	X	X	X	X	X	X	X	X	X		X	X
Torus			X	X		X					X				X
More complex topologies			X	X							X	X		X	
Concentrated topologies											X	X		X	
Number of cores															
4													X		
8						X							X		
16	X	X		X	X	X	X	X					X		X
32												X			
64			X				X		X	X	X	X		X	X
128												X			
Number of threads per core															
1			X		X	X		X	X	X		X	X		X
2												X			
4												X			
Memory subsystem															
Shared L2 cache			X			X		X					X		X
Private L2 cache					X				X	X			X		
3 cache levels					X				X	X		X			
Directory-based coherence			X		X	X			X	X		X			X
Broadcast-based coherence								X	X						

	References														
	[12]	[44]	[34]	[25]	[48]	[39]	[28]	[2]	[26]	[6]	[3]	[37]	[27]	[19]	This work
Scope															
Interconnect performance	X	X	X	X	X	X	X	X	X		X	X		X	X
System performance			X			X		X	X	X	X		X		X
Interconnect Power			X		X	X	X	X	X	X	X	X	X		X
Interconnect Area	X		X		X	X	X	X	X		X	X	X		X
Cache Power												X	X		X
Cache Area												X	X		X

Chapter 3
CMP Architecture Framework

This chapter presents the architecture of the chip multiprocessor we are modeling, including a more detailed description of the interconnection network.

3.1. General Description of the Architecture

This study focuses on homogeneous chip multiprocessors (CMPs). The system is composed of several *tiles* connected by an interconnection network. On each tile we have a core with a private first level cache (L1) split into data and instructions, a bank of the shared second level cache (L2), a router and we may also have a memory controller. Figure 3.1 depicts the block diagram of the chip and a tile with memory controller. It also includes the connections between the elements in the tile and the router. This router has two inputs and two outputs to connect the tile to other neighbouring ones.

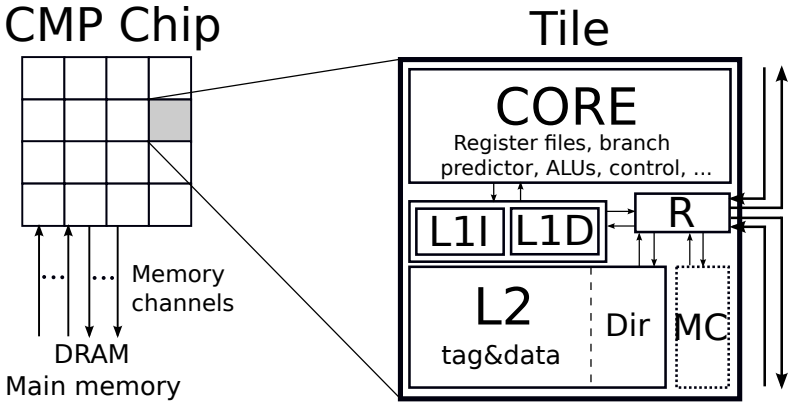


Figure 3.1: Block diagram including a chip and the components of a tile. MC stands for memory controller, R is the router and Dir is the directory, which is included in the L2 cache. This router has two ports to connect the tile to other neighbouring ones.

We are modeling a system with simple Ultrasparc III Plus single-thread in-order cores. They are running Solaris 10 operating system and execute one instruction at a time. We include systems with 16 and 64 cores implemented in 32nm technology. Table 3.1 summarizes the key parameters of our system. To model the architecture we based our design in other systems with similar characteristics, both from academia research papers [49, 39, 4] and commercial processors such as IBM Power4 [42], Tilera's TILE*Pro*64 [43], Intel 48-core processor [20, 21] and Sun Microsystems' Niagara2 [36].

Table 3.1: Main characteristics of the CMP system.

Cores	16 and 64 cores, Ultrasparc III Plus, in order, 1 instruction/cycle, single threaded, 2GHz frequency
Coherence protocol	Directory-based, MESI Directory distributed among L2 cache banks
Consistency model	Sequential
L1 cache	32KB data and instruction caches, 4-way set associative, 2-cycle hit access time, 64B line size Pseudo-LRU replacement policy
L2 cache	Distributed, 1 bank/tile, 1MB per bank, 16-way set associative, 64B line size Pseudo-LRU replacement policy shared, inclusive, interleaved by line address 7-cycle hit access time
Memory	4 memory controllers, distributed in the edges of the chip (both for 16 and 64-core architectures) 160-cycle latency

Chip multiprocessors are used to execute parallel applications (to reduce execution time) or independent programs on each core (to maximize throughput). In both cases, we need a memory coherence protocol. For parallel applications, it is used to maintain coherence among shared variables. For the independent programs, to allow migration of processes between cores. We are using a directory-based MESI coherence protocol. A MESI protocol has four states that depend on the coherence properties of the corresponding cache line: modified (M), exclusive (E), shared (S) and invalid (I).

All the traffic that traverses the interconnection network is a direct consequence of the memory activity, either to move cache lines (instructions or data) to the tile that needs them or for coherence management. That is why it is important to model the caches realistically, even though our main interest is the interconnect [27, 37].

The size of the messages traversing the network depends on the type of message and the size of the cache line (in our case, 64B). Control messages only need 8B for the address and protocol information, while data messages need those 8B plus 64B to hold the data. The assignment of resources in the interconnect depends on the type of the message, which is determined by the protocol. For that reason, the balance in the use of resources in the network depends on the type of traffic managed by the memory system. Also, the spatial distribution of the traffic depends on the location of the data in the chip, which is derived from the applications being executed and the interleaving of the data in the shared L2 cache banks.

Additional information about the CMP architecture and the memory hierarchy can be found in the Appendix B. It also includes a detailed description of the coherence protocols for both the L1 and L2 caches.

3.2. Interconnection Network

Although we carefully model every element of the system, we especially focus on the interconnection network. This element of the system is the responsible for connecting all the tiles in the chip, which is why it is also called *network on chip* (NoC). In this section we include explanations about the topologies, the router architecture (including routing and flow control) and some notes about deadlock avoidance.

3.2.1. Topologies

We are going to study three different topologies: mesh, torus and ring, which are depicted in Figure 3.2. The *2D mesh* is a widespread choice for large-scale CMPs due to its regularity. The tiles are organized in a matrix and every node is connected to its four neighbouring nodes to the north, south, east and west. A *torus* is a mesh in which we add wraparound links to reduce the average number of hops between tiles. To avoid having a very long wraparound link, which would involve having links with different latency, the torus is *folded* so that every link is the same length, equal to the length of the mesh link multiplied by $\sqrt{2}$ [13]. Longer links involve higher wiring capacitance, resistance and latency [45]. This topology is the one that will need the most resources in area and power. In contrast, we have included a *bidirectional ring*. So as to keep the same organization of the chip, which follows a matrix layout, the ring is built as a hamiltonian cycle.

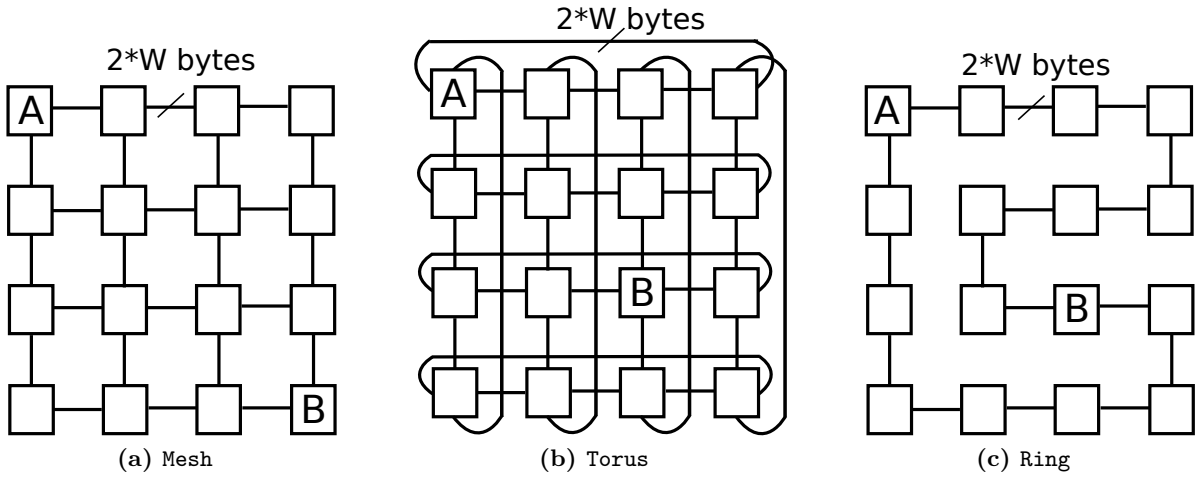


Figure 3.2: Diagrams of the topologies analysed in this project for a 16-core CMP. Every line represents two links, one on each direction. W is the link bandwidth. A message going from A to B would be traversing the maximum distance on each topology.

Table 3.2 summarizes the main characteristics of the three topologies. The number of input and output ports of the router is a direct indicator of the complexity; the higher the number of ports, the higher the area and expended energy. If we divided the network in two equal parts, the bandwidth we would have between the two parts is what we call the *bisection bandwidth*. A lower bisection bandwidth indicates that communications in the network will be slower. A *hop* in the network is a set of router and link the message traverses when going from source to destination. When counting the total number of hops we also include the first link going from the tile to the router. The number of hops gives us an idea of the time it will take a message to traverse the network. In the table, we distinguish the maximum distance (also called *diameter*) and the

average distance. A messages flowing from node A to B in Figure 3.2 would be traversing the maximum distance. In the hop count, we are including the hops between the tiles and the first and last routers. The length of the link will have an impact on the power consumed by the network.

Table 3.2: Qualitative comparison of the three topologies for a CMP system with N tiles (we assume that N will always be a perfect square). The number of inputs\outputs does not consider tiles with a memory controller, where routers would have one more input and output, or the tiles in the edges of the mesh, where some ports would be left unused. W is the link bandwidth and L is the wire length.

Topology	**inputs/ outputs**	**Bisection BW**	**Max. hops (diameter)**	**Avg. hops (Avg distance)**	**Link length**
2D mesh	6/6	$2W\sqrt{N}$	$2\sqrt{N}$	$2/3\sqrt{N}+1$	L
Torus	6/6	$8W\sqrt{N}$	$\sqrt{N}+2$	$1/2\sqrt{N}+2$	$L\sqrt{2}$
Ring	4/4	$4W$	$N/2+2$	$N/4$	L

3.2.2. Router Architecture

We use a pipelined router with four stages: input buffering and routing, virtual channel allocation, switch allocation and switch traversal. Therefore, every hop takes a total of five network cycles, including link traversal. Figure 3.3 illustrates the microarchitecture of a classic four-stage virtual channel pipelined router. At each input, there are several *virtual networks* (VNs), used to avoid protocol *deadlock* [13]. Deadlock occurs when messages are stalled in the network waiting for resources that are being held by other stalled messages, creating a cyclic dependency. A message is assigned to a VN depending on its class, which is determined by the memory coherence protocol. Our architecture has two virtual networks, so messages will be assigned to one or the other depending on whether they are a request or a reply. For each virtual network, we may have several *virtual channels* (VCs), each of them with a buffer to store incoming packets. VCs are used to prevent a packet that could continue traversing the network from being stalled after another that has been blocked [13]. Our routers have a total of four VCs per input, distributed in the two VNs. The network runs at 2GHz, using the same clock frequency as the processors. Table 3.3 sums up the main parameters of our interconnection network and routers.

A message flowing through the network is divided into *flow control units* (called *flits*) [13]. In our case, each flit is 16 bytes long, which is the same as the wire bandwidth. Control messages will need a single flit, while data messages will be divided in five flits. Figure 3.4 shows how a message composed of three flits would traverse the network through three routers. We are studying packet-switched networks, which means packets reserve resources as the advance through the network.

When the head flit of a message first arrives at a router, it is stored at the input buffer assigned to its virtual channel. In the same cycle, routing computation is performed, which indicates the output port through which the message will exit the router. We are using deterministic dimension order routing (DOR), also called X-Y routing. Messages are first routed on the X dimension and then, on the Y dimension. This means that all packets with the same source and destination will follow the same minimal path.

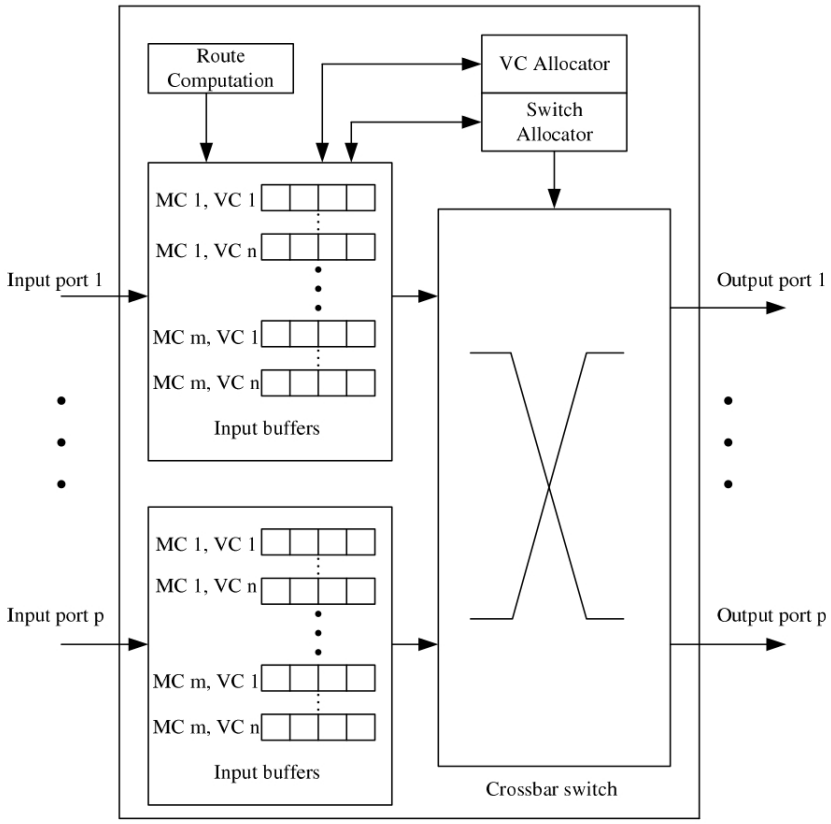


Figure 3.3: Four-stage virtual channel router. MC stands for message class, which is equivalent to virtual network. (Image taken from [50])

Table 3.3: Main characteristics of the interconnection network.

General	Two virtual networks (requests and replies)
Routers	4-stage pipeline: routing and input buffering, VC allocation, switch allocation and switch traversal Round-robin 2-phase VC/switch allocators 2 VCs per virtual network 5-flit buffers per virtual channel, enough to store a whole message (3-flits per buffer in ring with higher bandwidth)
Links	16-byte flit size (we also include a ring with higher bandwidth with 24B flit size) 1-cycle latency
Technology	32nm, 2GHz frequency, $V_{dd} = 1V$

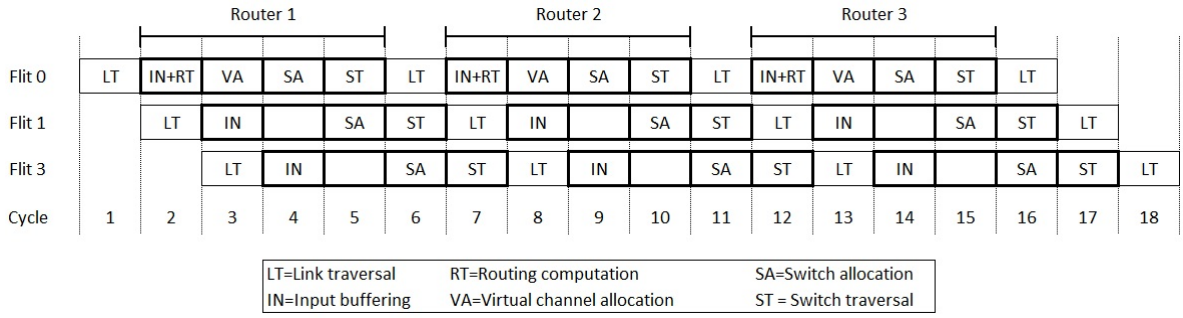


Figure 3.4: Chronogram of a message with three flits travelling through three 4-stage routers of the network

After that, a virtual channel must be allocated in the downstream router. In our implementation, virtual channel allocation has two phases. During the first phase, a destination VC is chosen for each VC at each input port. At the end of this phase, we will have a set of VCs at the downstream routers that have been requested by a set of competing VCs at the input ports. During the second phase, arbitration is performed to choose a single input VC channel for each VC at the downstream routers following round-robin order.

To leave the router, the flit must traverse the crossbar, which connects all inputs with all outputs. There may be conflicting requests, so there is a two-phase switch allocation stage. During the first phase, a single VC is chosen for each input port. In the second phase, arbitration is performed so that only one request is granted for each output port. After this, flits may traverse the crossbar, exit the router, traverse the link, and get to the next router.

Since all flits that belong to the same message always follow the same route and use the same virtual channels, routing and VC allocation must only be performed for the head flit. A VC in a router may only be used by one message at a time. It will only be available for another message to use when the tail flit of the message has left the router.

Our network uses *wormhole credit-based* flow control. With wormhole flow control, flits can exit a router as soon as they arrive and resources are allocated in flit-granularity. This is a better implementation than *store-and-forward*, in which we have to wait for a whole packet to arrive before we can forward it to the next router, and *virtual cut-through*, where we need to reserve space for the whole packet as soon as the first flit arrives. Credit-based flow control refers to how the router knows that it can allocate a VC in the downstream router because there is free buffer space. The upstream router simply keeps a count of the number of free flit buffers in each downstream VC and decreases the count every time it forwards a flit. When a flit buffer is freed in the downstream buffer, it sends a *credit* to the upstream router, so that it can increment its buffer count.

We have been talking about a fixed four-stage router, but we also want to be able to experiment with routers that could be traversed in less than four network cycles. This can be useful for merging stages to decrease latency when we are working at a reduced frequency and voltage for power saving [50]. Also, in the ring topology, the number of inputs/outputs to the outside of the tile is reduced to two (as opposed to the four used in mesh and tours), which results in a smaller number of buffers and simpler allocators and crossbar. For this reason, since the complexity of the crossbar is considerably reduced, we have also included tests merging the switch allocation and switch traversal stages. The diagram of a 3-flit packet traversing three of

these reduced-pipeline routers is shown in Figure 3.5.

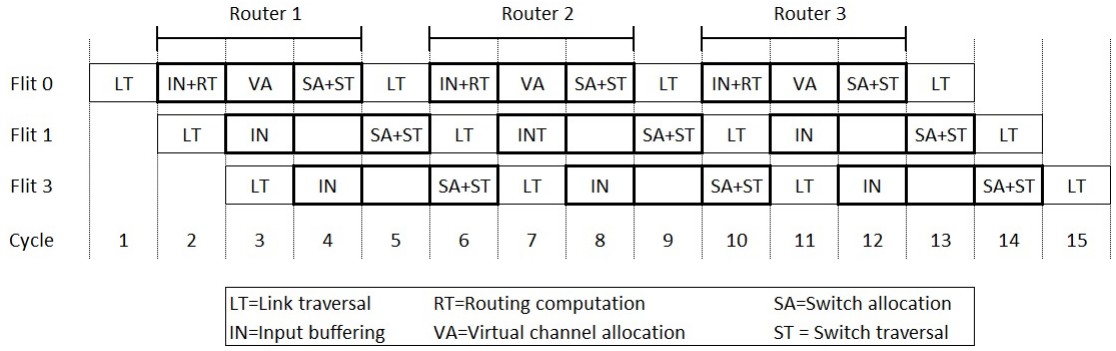


Figure 3.5: Chronogram of a message with three flits travelling through three 3-stage routers of the network

Following the same idea, the routers in the ring topology will need a much smaller area. To make use of this idle space, we have also tested a configuration in which we increase the link bandwidth keeping the router area in the ring slightly under the router area in the torus. This has allowed us to have flits of 24 bytes, which will reduce the number of flits needed per message and, therefore, serialization latency.

3.2.3. Deadlock Avoidance

We have already mentioned that we use two separate virtual networks (one for requests and one for replies) to avoid protocol deadlock, but there are other causes for deadlock that we must take into consideration.

With the torus and ring topologies, the links form loops which could cause deadlocks. We can easily imagine all tiles in a loop sending a message to a tile a couple hops further away in the loop, booking resources and being stalled in a cyclic dependency. To avoid these deadlocks caused by loops in the topology we need two virtual channels per virtual network [14, 13]. The main idea is to choose a point in the loop, which we will call *dateline*, and force all flits to use either the first virtual channel or the second one depending on whether they are going to cross the dateline or not [14, 15, 38]. This method changes the original round-robin policy mentioned earlier. We route flits that will cross the dateline in VC1 and move them to VC0 after they have crossed. For all other flits, we use VC0. This way, we eliminate the loops in the topology and therefore, avoid deadlock. In this naive implementation, most flits will be routed on VC0, so the use of the virtual channels will be highly unbalanced. To achieve a balanced use of resources, we only force VC1 when the flit will cross the dateline. In any other case, we choose the virtual channel using round-robin.

Apart from that, in our tests we saw deadlock happening in some particular cases where there was a high congestion in the network. After analysing the debug traces produced by the simulator, we realized that it was due to some coherence messages overtaking other previous ones targeting the same cache line. To solve the problem, we tried using three virtual networks, one exclusively dedicated to these high priority messages that should always arrive sooner. This solved the problem, but forced us to have routers with six virtual channels, which was not our desired configuration. We finally decided to include a partial ordering in the interconnection network. A message will not be able to go past VC allocation or switch allocation if there is another message with higher priority waiting to get to same cache line. This also solved the

problem and allowed us to keep only four virtual channels.

Chapter 4
Methodology

In this chapter we introduce the metrics, workloads and simulation environment used in the project. More information about these topics can be found in Appendix C

4.1. Metrics

Our study focuses on the comparison of interconnection network topologies. Therefore, we are using several traditional interconnect-centric metrics. We study node throughput, link utilization, hop count, latency, virtual channel load and the number of arbitration requests. These metrics help us gain insight on how each topology behaves with the workloads.

Power consumption is a key factor in the design of new architectures. We include the energy and area costs of the interconnect and compare them to those of the memory subsystem.

We are modeling all the components of the architecture and we want to examine the impact the different network configurations have on the whole system, so we include metrics that that reflect the overall performance and the behaviour of the memory hierarchy. We analyse the execution time of every application with all the configurations to see which one leads to better performance. We also include miss rate results, which are a determinant factor in the pressure imposed on the interconnect.

4.2. Workloads

The chip multiprocessors we are focusing on may be used to execute parallel applications in order to reduce execution time or for multiprogrammed workloads (execution of independent programs on each core), to increase throughput. We are using a selection of shared-memory parallel applications from PARSEC and SPLASH2 and a multiprogrammed workload made up of SPEC CPU2006 benchmarks. The performance of these applications may be limited by the interconnection network and we intend to establish to what extent this is actually happening. We would like to work with pressing applications that offered a very high miss rate, a large amount of traffic between nodes and good scaling. To be fair, we tried to put together a set of applications with varying characteristics in terms of the axes previously mentioned. More information about the characteristics of the benchmarks and the selection process can be found in Appendix C.

PARSEC is a benchmark suite composed of multithreaded programs implemented with OpenMP and pthreads [11, 9, 10, 5, 7, 8]. It focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. We

performed a complete study of this benchmark suite and presented our results in a poster session at the HiPEAC international conference in January 2012 [32]. The benchmarks chosen are `blackscholes`, `canneal`, `fluidanimate`, `swaptions` and `x264`. They have all been executed with the large input except for `x264`, for which we have used the medium input due the large simulation time. We simulate the whole parallel region of the applications.

SPLASH2 is a mature benchmark suite containing a variety of high performance computing and graphics applications [46, 40, 10, 5]. Shared-memory parallelism is explicitly written by using PARMACS, a library of parallel macros that allow an architecture-independent implementation [29, 30]. The chosen applications are `barnes`, `fmm`, `ocean`, `radiosity`, `volrend`, and `water-spatial`. As we do with PARSEC, we simulate the parallel region of the applications.

SPEC CPU2006 is a benchmark suite composed of single threaded applications written in C, C++ and Fortran [41]. We have used it to build a multiprogrammed workload in which we run one application on each core, binding applications to cores so that no migration occurs. The threads that are being executed in each core are independent, so the only traffic in the network will be caused by cache misses and replacements. Since no data will be shared and no migration is allowed, there will be no additional coherency messages. In this case, we chose 16 applications with high footprint and working set size (according to [18, 16]) so as to determine potential bottlenecks in the interconnect. To build the workload for the 16-core architectures we have executed each application once, binding each one of them to a different core. For the 64-core architectures we have used the applications four times assigning them to the cores consecutively (this should have better been done randomly, that is left for future work). To execute this workload, we first warm up the caches for 200 million cycles and then execute for 500 million cycles.

4.3. Simulation Environment

We have obtained our results using simulation, which is a common resource in computer architecture for the exploration of the design space of new computing systems. We have chosen a set of tools used extensively in this field. We are using Wind River Simics as a simulator [31] and GEMS modules, from the University of Wisconsin [33]. Simics can simulate a full-system (which can execute an operating system), both single and multiprocessors, and it is widely used in this field. GEMS provides modules to model the memory subsystem (Ruby). GARNET, implemented in Princeton University and included inside Ruby, simulates the details of the interconnection network [1].

We carefully model all the components of the system and perform full-system simulation with simple single-threaded cores and directory-based coherence. We have had to include several modifications to the Ruby module to produce the statistics and support the topologies we wanted to study. For the power and area analysis we used two additional tools, CACTI [35] and Orion [23, 24]. We describe this in detail in the following sections.

New Functionalities Incorporated into the Simulator

To model the system with our desired features we had to include the following features in the simulator:

- Modeling of the three topologies (mesh, torus and ring) using *network files*. With these files, we simply need to indicate which elements to include in each node and how to connect them.
- Modify the protocol to use only two virtual networks, as opposed to the original five. Request messages will be assigned to one of them, and replies to the other.
- Prevent torus and ring from using some minimal paths more than others. If there are two minimal paths between two nodes (this happens in every loop with an even number of nodes), the same one is always selected in detriment of the other, depending on which link was written first in the network file. To avoid this, we statically force the minimal path to be only one of the two, alternating the direction the messages will follow depending on whether the source node identifier is even or odd [38].
- Deadlock avoidance implementation for torus and ring, as explained in Chapter 3.
- Solve deadlock caused by coherence messages overtaking other previous ones targeting the same cache line, explained in Chapter 3. We consider acknowledgements as high priority messages, because they mean there is a cache line being blocked waiting for them. We implemented another version of the protocol with three virtual networks, one exclusively dedicated to these high priority messages. This solved the problem, but forced us to have routers with six virtual channels, which was not our desired configuration. We also implemented a partial ordering in the interconnection network, so that no message could move forward if there is another one with higher priority waiting to go to the same cache line.
- Implementation of the capability of combining any consecutive stages of the router pipeline so that they can be executed in the same cycle. This allows us to end up with a one-cycle router. To implement this, we included the possibility of scheduling an event (the wake up of the next stage) for the current cycle and introduced an ordering of the events waiting to be scheduled.

GARNET offers statistics for average link utilization, virtual channel load and network latency. We included the following additional measurements:

- Number of injected and received messages per network interface (each L1, L2 block and memory controller).
- Detailed information about the utilization of each link.
- Number of occurrences for message latency and hop count.
- Number of accesses and misses in the L1 and L2 for power analysis (this was already implemented but had to be fixed).
- Number of messages sent by caches and memory controllers broken into coherence types.
- Number of times virtual channel and switch allocation are requested but arbitration fails.

Power and Area Analysis

To get the area and the energy expended by the network we used a circuit modeling tool called Orion 2.0 [23, 24]. The version of Orion included in Garnet was 1.0, which didn't give accurate results compared to the new version. The first thing we did was to integrate the new

version in our Ruby module. Orion 2.0 gives us information about the area, leakage and dynamic power for the links, the router stages (input, virtual channel allocation, switch allocation and crossbar traversal) and the propagation of the clock.

For the area and energy expended by the memory hierarchy we used CACTI 6.5, which is another circuit modeling tool for estimating access time, cycle time, area, leakage, and dynamic power [35]. For the dynamic energy, CACTI tells us how much energy the cache consumes when performing a switching event, such as a tag read (TR), tag write (TW), data read (DR) and data write (DW). We then have to count the events that take place in the cache when we run the program (read hit, read miss, replacement...) and multiply them by their consumption.

The technology files in both tools have been matched so as to be able to compare the results obtained from the two models. These files specify parameters such as the technology, voltage, transistor sizes, resistivities, capacitances, wire sizes,... Some parameters have been taken from the work of Gracia *et al.* [17] and accuracy has been further improved by approximating our values to the ones used in INTEL 32nm technologies [22].

Chapter 5
Main Results

This chapter summarizes the main contributions of our analysis for 16 and 64-core architectures, including both parallel and multiprogrammed workloads. We focus on the comparison of performance, power, area and traffic distribution for the mesh, torus and ring topologies. We include a ring with increased bandwidth and one with 3-cycle routers. More detailed information is included in Appendix D

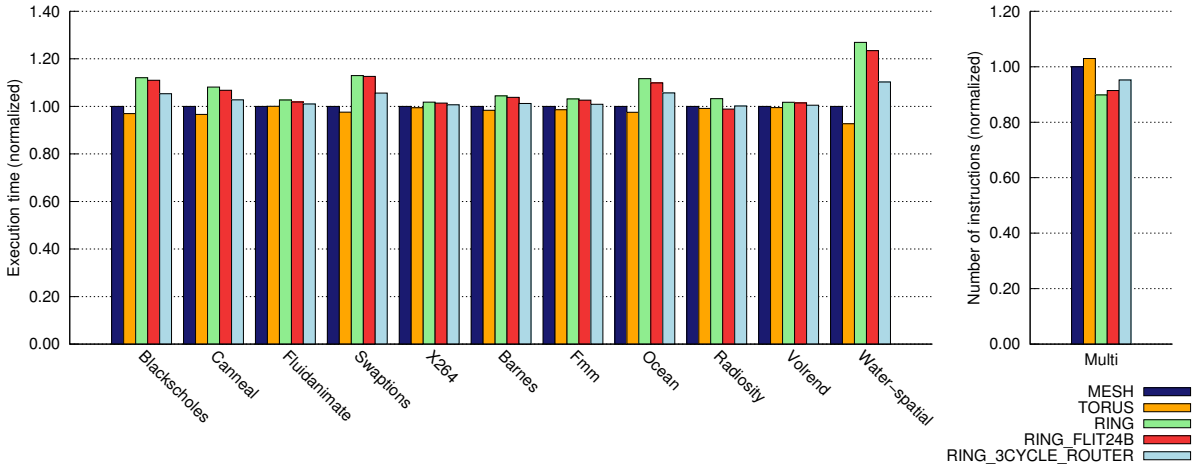
5.1. Network Topology Comparison

Our objective is to compare the behaviour of the ring, mesh and torus topologies. The ring is a simple topology that is expected to have worse performance but less power and area costs. The torus is the most complex topology out of all three. It has more links that provide lower communication latency but suffers from higher energy consumption. The mesh is an intermediate option used very frequently due to its regularity. To benefit from the small area of the original ring topology, two additional rings have been included: one with higher bandwidth where flits are 24 bytes long (instead of the original 16 bytes), and one with 3-cycle routers (instead of 4-cycle routers).

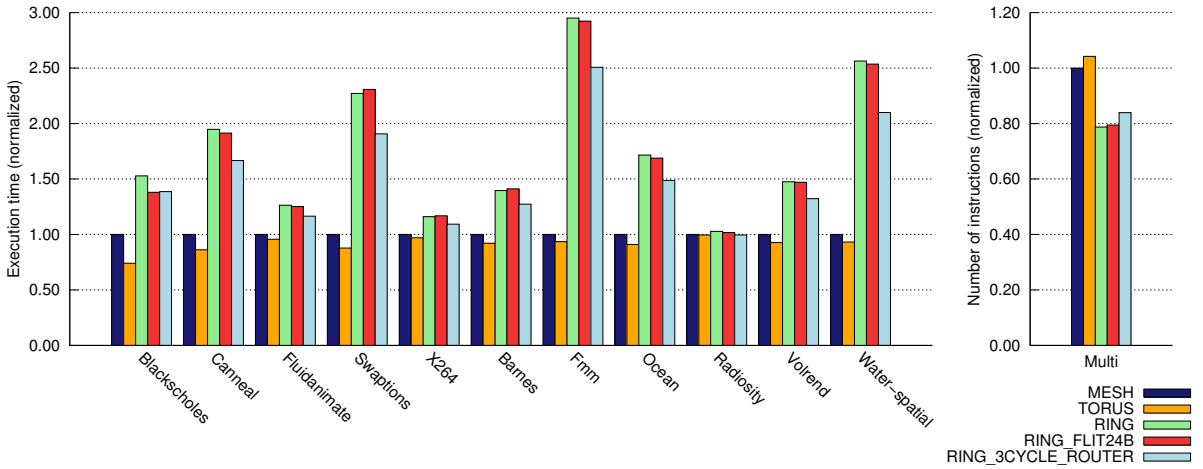
In this section, we study the performance of the applications (both parallel and multipgrogrammed workloads) with all the network configurations. We also include power and area costs and analyze the tradeoffs for each topology.

5.1.1. Performance

To compare the impact of the network configurations on performance, we are studying the number of processor cycles it takes for the parallel workloads to complete the parallel section. For the multiprogrammed workloads, we check how many instructions get executed in 500 million cycles. Figure 5.1 shows these values normalized to the mesh topology. We can see that we achieve the best performance with the torus topology, closely followed by the mesh. In 16-core architectures differences between topologies are much smaller, with the ring with 3-cycle routers being very similar to the mesh for some applications. In 64-core applications, the performance of the ring topologies drops significantly. The ring with higher bandwidth (with 24B flits and links, as opposed to the standard 16B) performs only slightly better than the original ring, which does not compensate for the increase in area and power (as we will demonstrate in the next section). The ring with 3-cycle routers instead of 4-cycle ones also performs better than the original ring, but is still much worse than the mesh and torus.



(a) 16 cores



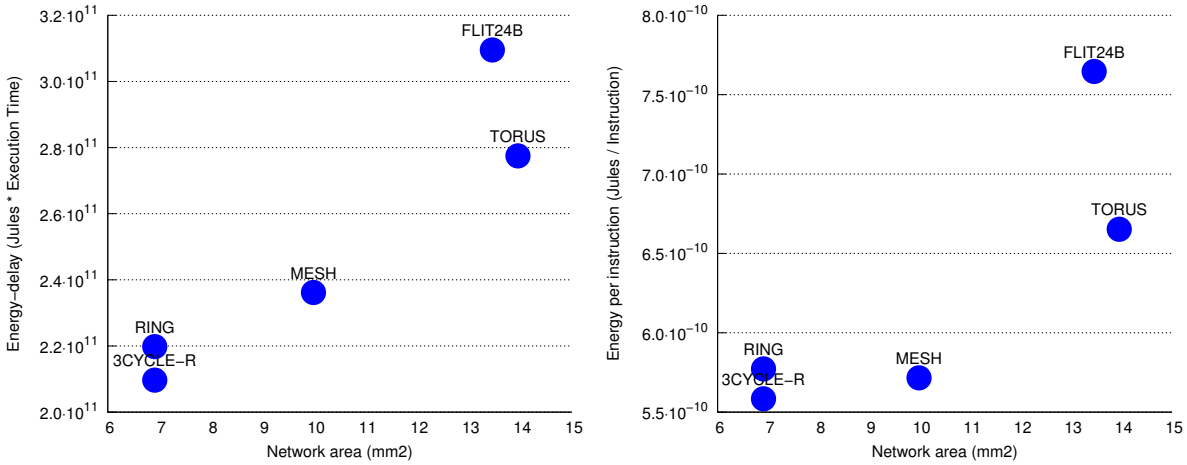
(b) 64 cores

Figure 5.1: Performance normalized to the mesh topology for 16 and 64 cores. For parallel workloads, we measure the execution time of the parallel region. For multiprogrammed workloads, we count the number of completed instructions in 500 million cycles. The figures in the left include the paralell workloads; for those bars, the higher the better. The ones in the right show results for the multiprogrammed workloads; in this case, the lower the better. Note that scales are not consistent among graphs.

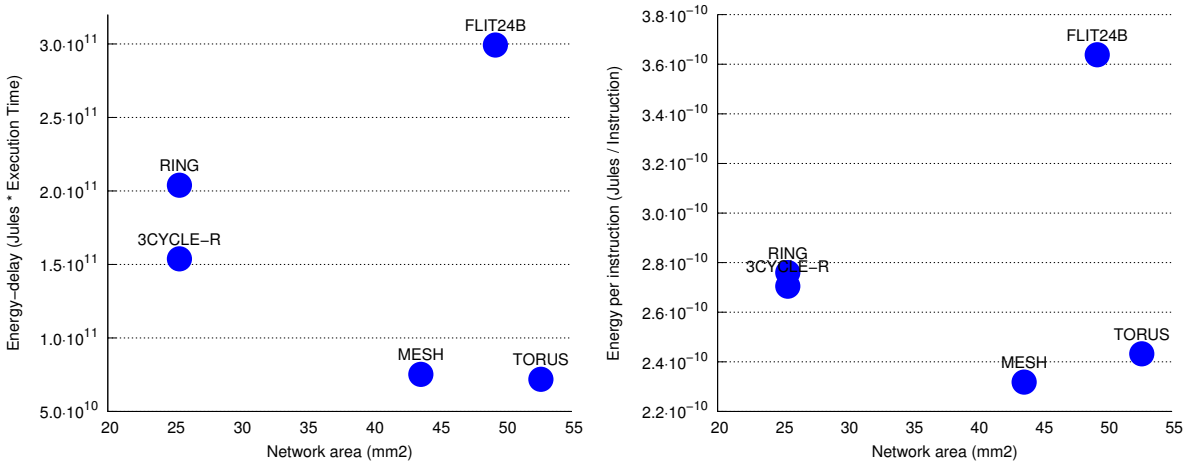
The differences in performance are a direct consequence of the number of hops it takes a message to go from its source to its destination. Besides, there is no congestion in the network that could be slowing messages down. These results ratify the conclusions of Sanchez *et al.* A more in depth explanation can be found in Section D.1.

5.1.2. Energy-Delay versus Area

When making design choices for future architectures we need to consider performance, power and area. When executing parallel applications on a multicore architecture, we are interested in reducing latency. Therefore, we are calculating energy-delay, which is commonly used to account for both energy consumption and performance improvements. This allows us to visualize the tradeoff and decide if the performance increase we are getting is worth the higher consumption. Using chip multiprocessors to run multiprogrammed workloads, we want to increase our throughput. For that reason, we are going to use energy per instruction (EPI), which indicates the



(a) Parallel applications (left) and multiprogrammed workloads (right) executed on 16 cores



(b) Parallel applications (left) and multiprogrammed workloads (right) executed on 64 cores

Figure 5.2: Area versus energy-delay or energy per instruction for 16 and 64 cores. We also distinguish between parallel applications (left) and multiprogrammed workloads (right). For the parallel applications, they have all been considered together, as if they were executed back to back.

quality of the system in terms of throughput by modeling how much energy is needed for the execution of each instruction. For these workloads, we ran the simulations for a fixed number of cycles (as we explained in Chapter 4), so the energy-delay metric would not be representative. We are considering only the energy expended by the interconnection network since it is the element we are comparing and we noticed that cache energy does not vary among interconnect configurations (see Section D.4).

Figure 5.2 represents area versus energy-delay or EPI for 16 and 64-core architectures, which will help us to make a decision based on the three main aspects we should consider. First, we are going to focus only on the vertical axis, which is representing the energy-delay or EPI. For parallel workloads, we are adding up the energy-delay values of all the applications as if they were all executed back to back. When using 16 cores, we are getting the best results with the ring topology with 3-cycle routers, which is similar to the original ring topology. However, if we focus on our 64-core architecture, we most often get the best results with the torus, very closely followed by the mesh. This is because, as we saw in Section 5.1.1, performance drops much more

significantly for the ring topologies when we have a higher core count due to the number of hops. In those cases, the increase in energy consumption is justified by the large benefits we get in performance.

We are now going to consider the area requirements for each topology, represented in the horizontal axis of Figure 5.2. The torus is the topology with larger area requirements and the ring the one with the lowest. The ring with higher bandwidth was designed so that routers would have the same area as the torus routers. The area for the ring with 3-cycle routers is the same as the area for the original ring.

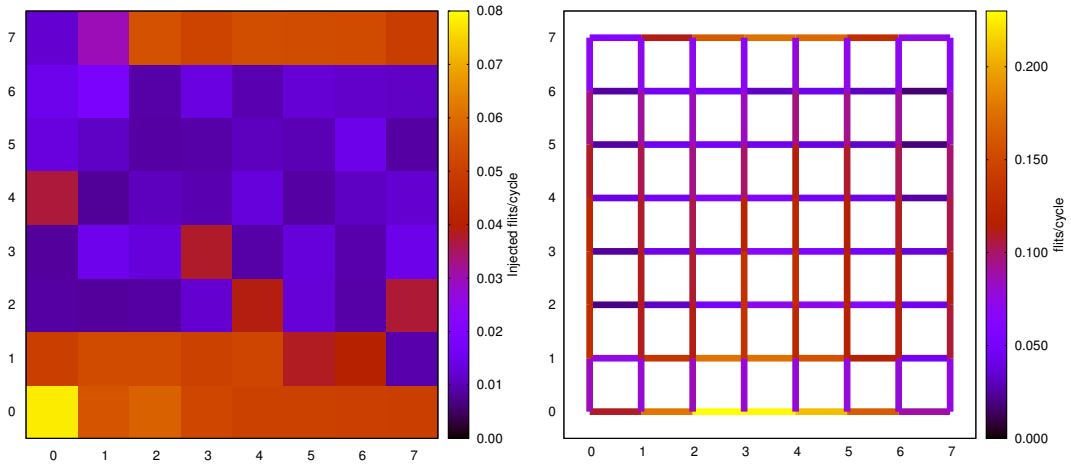
If we look at all the parameters included in the graph, we would like to have a configuration with small area and small energy-delay, that is, be in the bottom left corner of the graphs. For the 16-core system, we would clearly choose the ring topology with 3-cycle routers, which gave us the best energy-delay values and has only very small area. With 64 cores, we could draw a line between the 3-cycle router ring and the mesh topology to represent the Pareto optimal points. Anything above that line would be suboptimal. We see mesh and torus very close with the lowest energy-delay values, but the torus needs a much larger area. Ring topologies still have smaller area, but performance has dropped considerably. In this case, we would choose the mesh topology to get the best tradeoffs. Conclusions are applicable to both parallel and multiprogrammed workloads.

5.2. Non Uniform Traffic Distribution

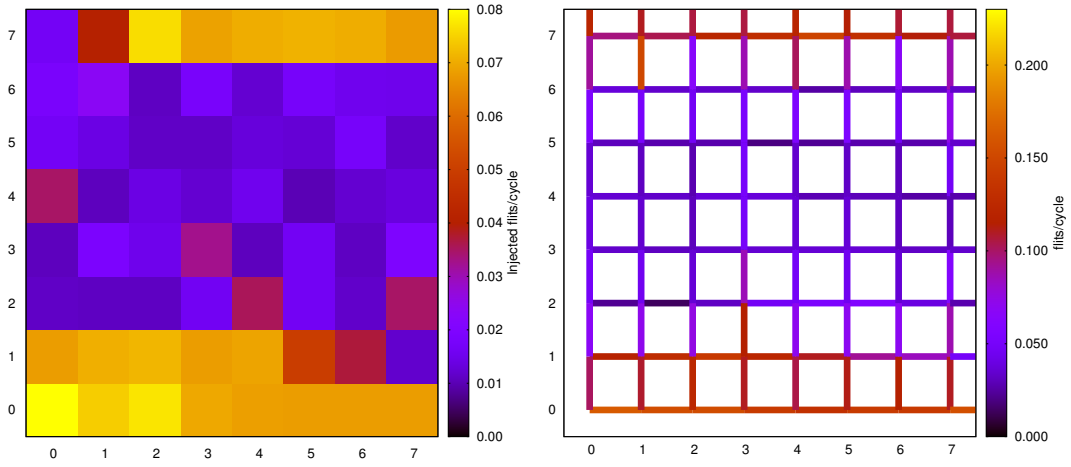
We have analysed the number of injected flits and the link utilization for all our architecture configurations and workloads. We have noted that traffic is unevenly distributed in the interconnect, which means that some resources will be needed more often than others. In this section, we are only presenting results for `blackscholes` among all the parallel applications. We have results for all the other applications but we will omit them due to space constraints. The same conclusions are extracted from all parallel applications. We will also be focusing on results for a 64-core chip, but conclusions still hold for 16-core configurations.

Figures 5.3 and 5.4 depict a heat map of injected flits per cycle for each node and link utilization for `blackscholes` executed on 64 cores. The distribution of injection flits is the same regardless of the topology. Values are usually smaller for the rings because a very similar amount of flits gets injected in a much longer period of time. Besides, for the ring with higher bandwidth (24B flits), flits are bigger so we need less flits to send the same amount of information. We can clearly see that some nodes inject more flits than others. Parallel workloads usually have a master thread that drives the execution and distributes work to other threads, which might not be used uniformly. In this case, judging from the heatmaps, we could say that the master thread is located in the bottom left corner of the chip. We can also see that link utilization is higher around the nodes with higher injection rates. Also, link utilization is higher in the ring topologies, since there are less links to transport the same amount of information. The torus wastes more resources since it is the topology where the highest number of flits get injected per cycle, but still has the lowest link usage. Even though we see different patterns in other applications, the conclusions we draw from them are exactly the same.

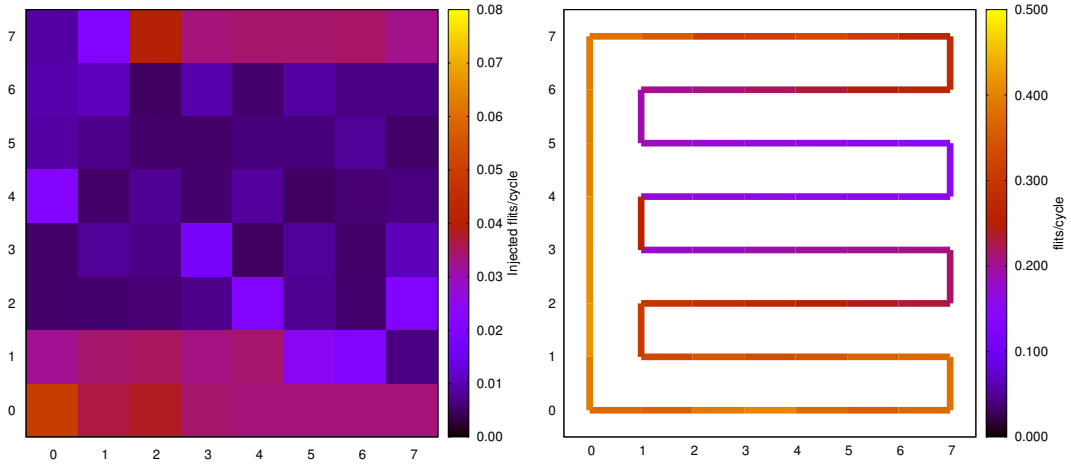
Figures 5.5 and 5.6 show the same plots for the execution of a multiprogrammed workload. In this case, we see four clear hotspots in the injection pattern in the edges of the chip. Those are the



(a) Mesh

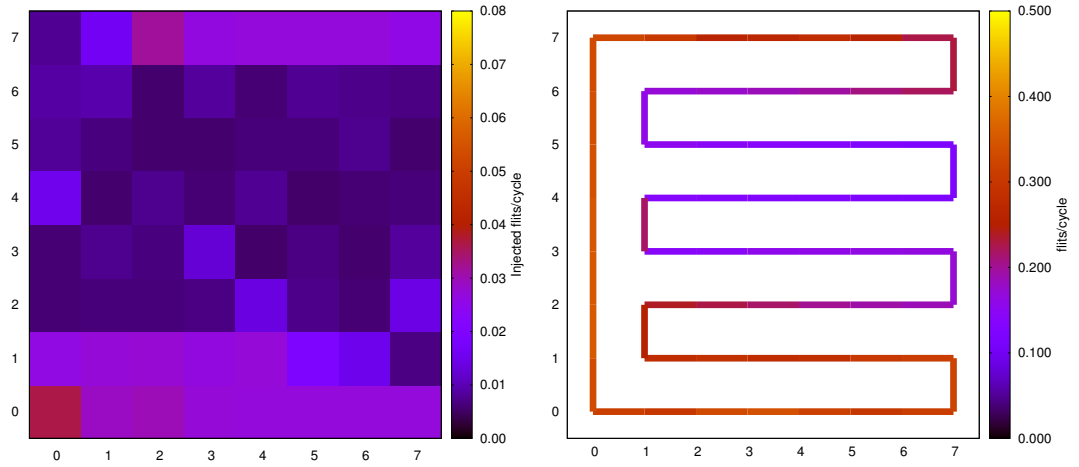


(b) Torus

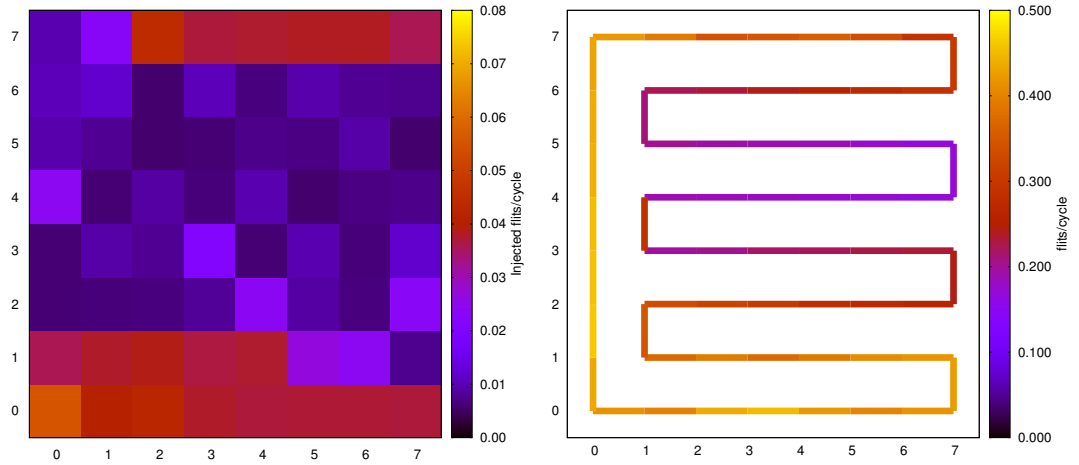


(c) Ring

Figure 5.3: Injected flits per node (left) and link utilization (right) for the `blackscholes` application executed in 64 cores. For link utilization, each line is the combination of two links, one in each direction. Injection and ejection links have been left out, since their usage can be easily infered from injection values. Note that the scale has been kept constant among topologies for injection figures, but not for link utilization. In the torus, links that touch the edges of the chip represent the wraparound links. (Part A)



(a) Ring, 24Bflits

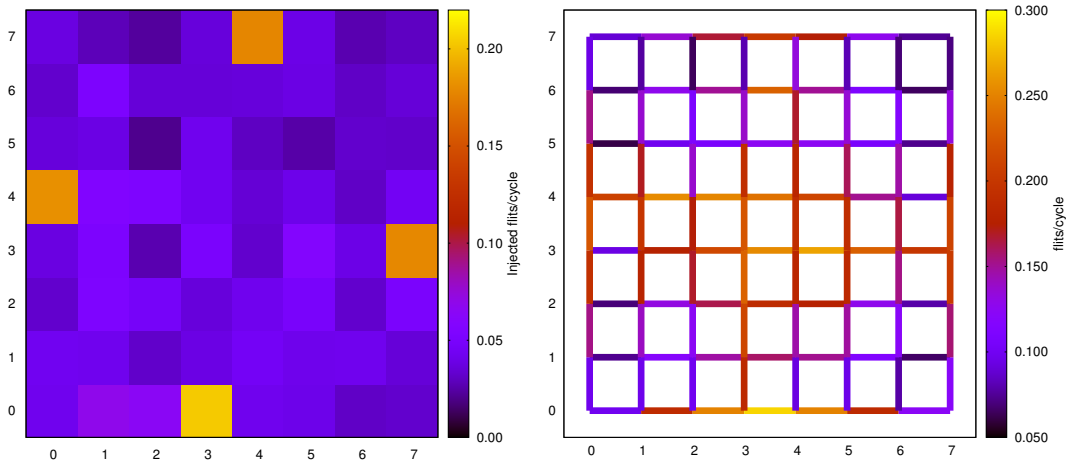


(b) Ring, 3-cycle routers

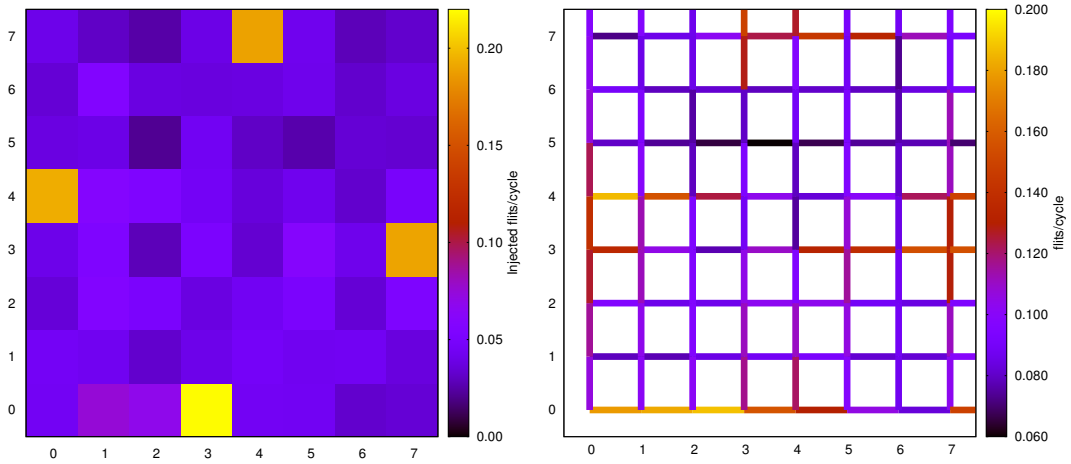
Figure 5.4: Injected flits per node (left) and link utilization (right) for the `blackscholes` application executed in 64 cores. (Part B)

tiles where the memory controllers are located. Apart from that, the rest of ideas we introduced for parallel workloads are still valid. In the mesh topology we can also see that links are more used in the center of the chip, which is the characteristic behaviour for this topology with uniform traffic. Probably, the memory controller location increases link usage in the center of the chip.

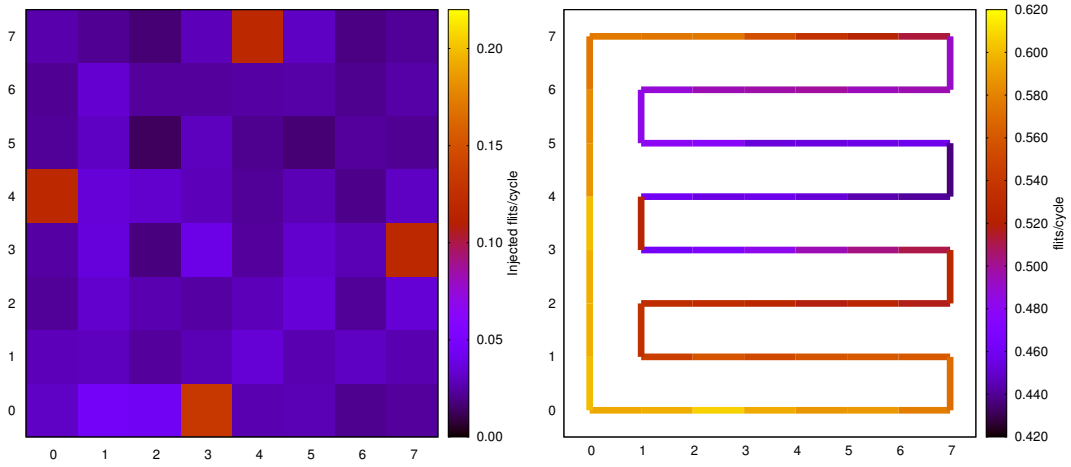
These results point out that the network is not being uniformly used. We could have an heterogeneous network where some tiles had more resources and move the threads that need them to those locations. That way, we would be saving power in the *lighter* parts of the chip and would not waste so many resources. Mishra *et al.* already introduce this idea, but they only apply it statically knowing that the center of the chip is usually used more often in mesh topologies [34]. As we have seen, this is not always the case. As far as we know, there is no previous research that introuduces the idea of non uniform traffic derived from the behaviour of applications.



(a) Mesh

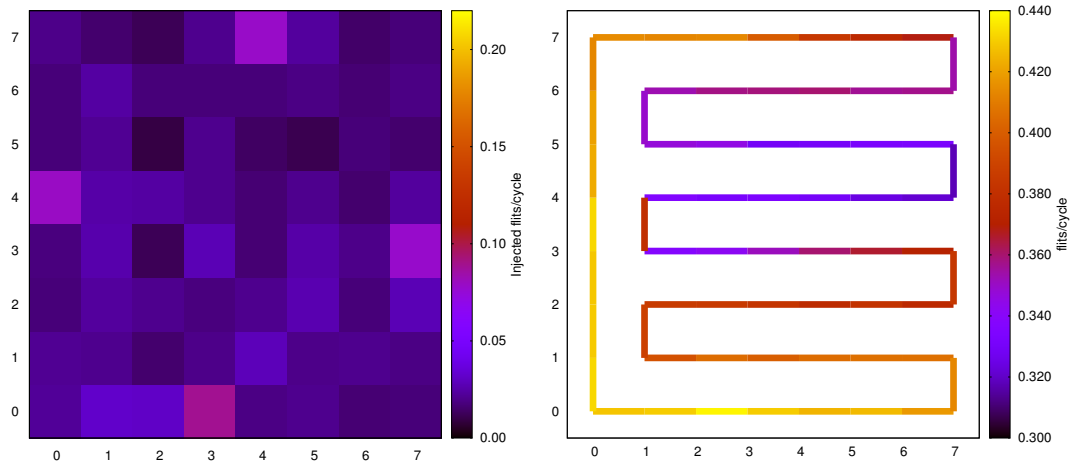


(b) Torus

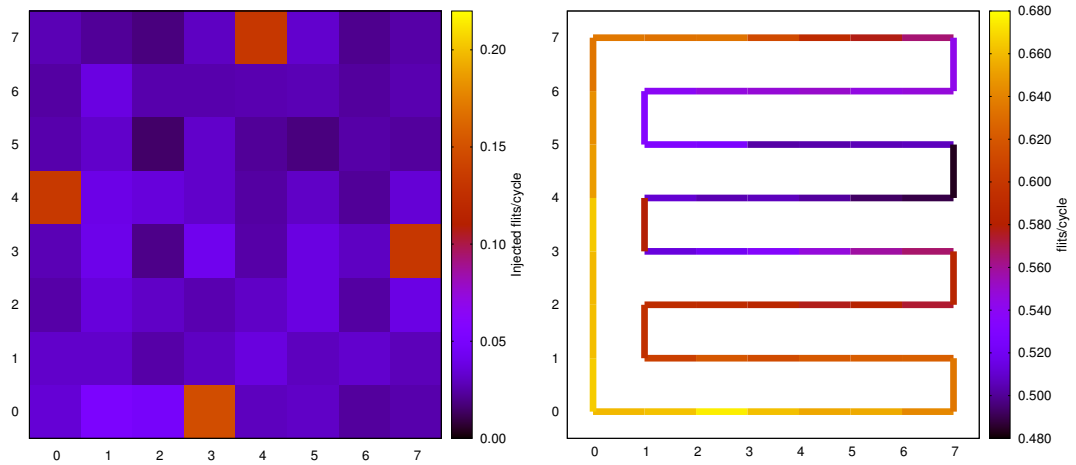


(c) Ring

Figure 5.5: Injected flits per node (left) and link utilization (right) for the multiprogrammed workload application executed in 64 cores. (Part A)



(a) Ring, 24Bflits



(b) Ring, 3-cycle routers

Figure 5.6: Injected flits per node (left) and link utilization (right) for the multiprogrammed workload application executed in 64 cores. (Part B)

Chapter 6
Conclusions and future work

In this chapter we sum up the conclusions we have taken away from this project and we introduce some ideas for our future research.

6.1. Conclusions

Interconnection networks have a significant influence on system performance, area and power consumption. The actual trend in computer architecture design consists on integrating several cores on a single chip. The network on chip is responsible for the communication between those cores and the caches, and its design is crucial for guaranteeing improvements in performance.

We have compared the behaviour of three network topologies: mesh, torus and ring. We include two additional ring configurations that benefit from the simplicity of the routers: one with higher bandwidth that uses bigger flits and, therefore, reduces serialization latency, and one with 3-cyle routers which will reduce the latency of every hop. In order to get representative results, we are carefully modeling the processors, memory hierarchy and the network, using full-system simulation for 16 and 64 cores. We execute real applications, including both parallel and multiprogrammed workloads. We performed a complete study of one of the benchmark suites featured in this project (PARSEC) and presented our results in a poster session at an international conference.

We have demonstrated that performance is highly affected by the choice of the interconnect in 64-core systems. The ring topologies produce much larger execution times due the increased number of hops it takes to traverse the network. The torus has the best performance, but with higher power and area costs. In this case, the mesh would be the best choice. On the other hand, for 16-core chips, differences in performances are not so big and a ring topology with 3-cycle routers offers acceptable performance with the lowest power consumption and area requirements.

Our most significant contribution is related to the distribution of traffic on the network. We have seen that traffic is not uniformly distributed on the network and that the tiles with higher injection rates vary with the applications. For multiprogrammed workloads, hotspots are always located in the memory controllers. As far as we know, there is no previous research where this behaviour has been noted.

We should take all this conclusions into consideration for the design of interconnection networks for new architectures.

6.2. Future Work

We have seen that the number of hops messages need to traverse the network has a large impact on performance. We need to reduce the hop count but we cannot do it at the expense of power costs. Increasing the number of resources will not lead to optimal configurations, since congestion in the interconnect is not an issue. We will test the performance with concentrated topologies, that is, connecting each router to more than one processing core. This will make routers more complex, with a higher number of inputs and outputs, but reduce the total number of routers needed and the hop count.

Also, there are many topologies between the minimal ring and the common mesh that still remain unexplored. It has already been detected that a ring topology with some random added links brings big performance gains with little power overhead, but they did not test the effect this new topology would have on overall system performance [25].

We intend to study the fairness for multiprogrammed workloads, which was not included in this work due to lack of time. We want to check if, when running independent applications, some of them progress more than others and monopolize the use of the interconnect. We will also analyse if the configuration of the network has an effect on fairness.

We have already discussed the importance of a careful codesign of all the elements of the system. That is why we plan to incorporate our knowledge of the behaviour of the memory subsystem into the design of the interconnect. For example, we know that every request will be followed by a reply, so we could send the head flit even before the cache produces the data to reserve the resources along the way. If we were using power-saving techniques that *switched off* part of the network when it was not being used, this head flit could also be used to *switch it on*.

We have also seen that the injection of packets is not done uniformly by all cores in both parallel and multiprogrammed workloads. We could build an heterogeneous topology and move threads that need more resources to the areas that have them.

Bibliography

[1] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33 –42, 2009.

[2] Niket Agarwal, Li-Shiuan Peh, and Niraj K. Jha. In-network coherence filtering: snoopy coherence without broadcasts. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 232–243, New York, NY, USA, 2009. ACM.

[3] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 187–198, New York, NY, USA, 2006. ACM.

[4] Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 282–293, New York, NY, USA, 2000. ACM.

[5] Nick Barrow-Williams, Christian Fensch, and Simon Moore. A communication characterization of splash-2 and parsec. In *Proceedings of the 2009 International Symposium on Workload Characterization*, October 2009.

[6] George B. P. Bezerra, Stephanie Forrest, and Payman Zarkesh-Ha. Reducing energy and increasing performance with traffic optimization in many-core systems. In *Proceedings of the System Level Interconnect Prediction Workshop*, SLIP '11, pages 3:1–3:7, Piscataway, NJ, USA, 2011. IEEE Press.

[7] Major Bhadauria, Vincent M. Weaver, and Sally A. McKee. Understanding parsec performance on contemporary cmps. In *Proceedings of the 2009 International Symposium on Workload Characterization*, October 2009.

[8] C. Bienia and Kai Li. Fidelity and scaling of the parsec benchmark inputs. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1 –10, dec. 2010.

[9] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[10] Christian Bienia, Sanjeev Kumar, and Kai Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Proceedings of the 2008 International Symposium on Workload Characterization*, September 2008.

[11] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[12] Everton Carara, Fernando Moraes, and Ney Calazans. Router architecture for high-performance nocs. In *Proceedings of the 20th annual conference on Integrated circuits and systems design*, SBCCI '07, pages 111–116, New York, NY, USA, 2007. ACM.

[13] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[14] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986. 10.1007/BF01660031.

[15] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *Computers, IEEE Transactions on*, C-36(5):547 –553, may 1987.

[16] Darryl Gove. Cpu2006 working set size. *SIGARCH Comput. Archit. News*, 35(1):90–96, March 2007.

[17] D.S. Gracia, G. Dimitrakopoulos, T.M. Arnal, M.G.H. Katevenis, and V.V. Yufera. Lp-nuca: Networks-in-cache for high-performance low-power embedded processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(8):1510 –1523, aug. 2012.

[18] John L. Henning. Spec cpu2006 memory footprint. *SIGARCH Comput. Archit. News*, 35(1):84–89, March 2007.

[19] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, pages 31–36, New York, NY, USA, 2010. ACM.

[20] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173 –183, jan. 2011.

[21] Intel. The scc platform overview. 2012.

[22] C.-H. Jan, M. Agostinelli, M. Buehler, Z.-P. Chen, S.-J. Choi, G. Curello, H. Deshpande, S. Gannavaram, W. Hafez, U. Jalan, M. Kang, P. Kolar, K. Komeyli, B. Landau, A. Lake, N. Lazo, S.-H. Lee, T. Leo, J. Lin, N. Lindert, S. Ma, L. McGill, C. Meining, A. Paliwal, J. Park, K. Phoa, I. Post, N. Pradhan, M. Prince, A. Rahman, J. Rizk, L. Rockford, G. Sacks, A. Schmitz, H. Tashiro, C. Tsai, P. Vandervoorn, J. Xu, L. Yang, J.-Y. Yeh, J. Yip, K. Zhang, Y. Zhang, and P. Bai. A 32nm soc platform technology with 2nd generation high-k/metal gate transistors optimized for ultra low power, high performance, and high density product applications. In *Electron Devices Meeting (IEDM), 2009 IEEE International*, pages 1 –4, dec. 2009.

[23] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. Orion 2.0: A power-area simulator for interconnection networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(1):191 –196, jan. 2012.

[24] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 423–428, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[25] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. A case for random shortcut topologies for hpc interconnects. In *Proceedings of the 39th International Symposium on Computer Architecture*, ISCA '12, pages 177–188, Piscataway, NJ, USA, 2012. IEEE Press.

[26] Tushar Krishna, Li-Shiuan Peh, Bradford M. Beckmann, and Steven K. Reinhardt. Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pages 71–82, New York, NY, USA, 2011. ACM.

[27] Rakesh Kumar, Victor Zyuban, and Dean M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 408–419, Washington, DC, USA, 2005. IEEE Computer Society.

[28] Mario Lodde, Toni Roca, and José Flich. Heterogeneous network design for effective support of invalidation-based coherency protocols. In *Proceedings of the 2012 Interconnection Network Architecture: On-Chip, Multi-Chip Workshop*, INA-OCMC '12, pages 1–4, New York, NY, USA, 2012. ACM.

[29] E. L. Lusk and R. A. Overbeek. Use of monitors in fortran: a tutorial on the barrier, self-scheduling do-loop, and askfor monitors. In *on Parallel MIMD computation: HEP supercomputer and its applications*, pages 367–411, Cambridge, MA, USA, 1985. Massachusetts Institute of Technology.

[30] Ewing Lusk, James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ross Overbeek, James Patterson, and Rick Stevens. *Portable programs for parallel processors.* Holt, Rinehart & Winston, Austin, TX, USA, 1988.

[31] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50 –58, feb 2002.

[32] María Villarroya Darío Suárez Víctor Viñals. Marta Ortín, Jorge Albericio. Behaviour characterization of the parsec benchmark suite in the processor's memory hierarchy, 2012. Poster presented in the 7th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC conference) that took place in Paris 23-24 January 2012.

[33] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33:92–99, November 2005.

[34] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. A case for heterogeneous on-chip interconnects for cmps. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 389–400, New York, NY, USA, 2011. ACM.

[35] Naveen Muralimanohar and Rajeev Balasubramonian. Cacti 6.0: A tool to model large caches.

[36] U.G. Nawathe, M. Hassan, K.C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *Solid-State Circuits, IEEE Journal of*, 43(1):6 –20, jan. 2008.

[37] Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis. An analysis of on-chip interconnection networks for large-scale chip multiprocessors. *ACM Trans. Archit. Code Optim.*, 7(1):4:1–4:28, May 2010.

[38] Steve Scott and Greg Thorson. Optimized routing in the cray t3d. In Kevin Bolding and Lawrence Snyder, editors, *Parallel Computer Routing and Communication*, volume 853 of *Lecture Notes in Computer Science*, pages 281–294. Springer Berlin / Heidelberg, 1994. 10.1007/3-540-58429-3-44.

[39] Ciprian Seiculescu, Stavros Volos, Naser Khosro Pour, Babak Falsafi, and Giovanni De Micheli. CCNoC: On-Chip Interconnects for Cache-Coherent Manycore Server Chips. In *Proceedings of the Workshop on Energy-Efficient Design (WEED 2011)*, 2011.

[40] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared-memory. *SIGARCH Comput. Archit. News*, 20:5–44, March 1992.

[41] Standard Performance Evaluation Corporation (SPEC). Spec cpu2006, 2006. `http://www.spec.org/cpu2006/` (Last access september 2012).

[42] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5 –25, jan. 2002.

[43] Tilera. Tilepro64. 2008. `http://www.tilera.com/products/processors/TILEPro_Family` (Last access september 2012).

[44] Isask'har Walter, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Access regulation to hot-modules in wormhole nocs. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pages 137–148, Washington, DC, USA, 2007. IEEE Computer Society.

[45] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[46] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd annual international symposium on Computer architecture*, ISCA '95, pages 24–36, New York, NY, USA, 1995. ACM.

[47] Marta Ortín Obón. Directores: María Villarroya y Darío Suárez. Caracterización del comportamiento de la suite parsec en la jerarquía de memoria del procesador, 2011. Proyecto Fin de Carrera.

[48] Young Jin Yoon, Nicola Concer, Michele Petracca, and Luca Carloni. Virtual channels vs. multiple physical networks: a comparative analysis. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 162–165, New York, NY, USA, 2010. ACM.

[49] Michael Zhang and Krste Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.

[50] Pingqiang Zhou, Jieming Yin, Antonia Zhai, and Sachin S. Sapatnekar. Noc frequency scaling with flexible-pipeline routers. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, pages 403–408, Piscataway, NJ, USA, 2011. IEEE Press.

Appendix A
Project Management

This appendix contains detail about time invested in this project and how it has been managed, as well as some problems that we have faced.

A.1. Time Management

This project has been developed between February and September of 2012, with full-time dedication. Figure A.1 depicts a Gantt diagram that shows how the tasks have been executed along these months.

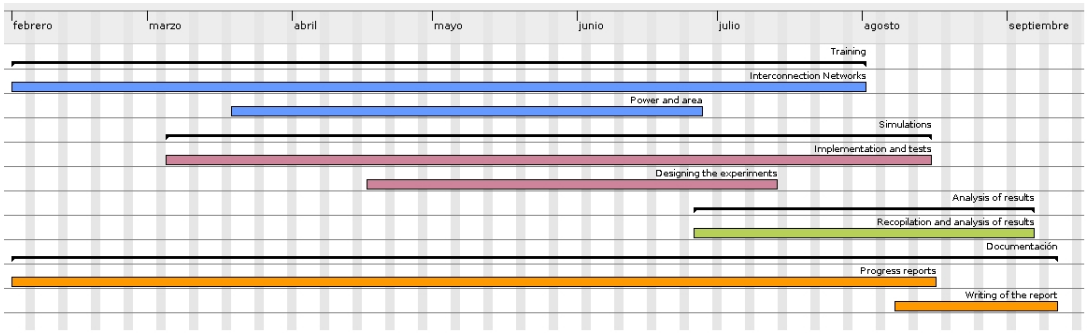


Figure A.1: Gantt diagram of the project.

In the following lines we include a short description of each task:

- **Training.** This task is extended throughout the whole project. We have mainly focused on the state of the art of interconnection networks by reading numerous papers and studying some books. We had already used the simulator adopted for the experiments, but have had to learn about power consumption and the tools for modeling area, power and delay.

- **Simulations**. This is the task that has taken most of the time. A very significant amount of hours has been invested on improving the simulator, implementing new features needed to model the architecture and testing. We also include in this section the design of the experiments we had to complete to get the results.

- **Analysis of results.** In this category we include the recopilation of results and analysis to validate or refute our hypothesis and extract conclusions.

- **Documentation.** During the project we have been periodically writing reports to document our progress and meetings. The writing of this report has taken most of the documentation time.

All the planned tasks were carried out during the development of this project and all the work was finished by the deadline.

A.2. Effort Invested in this Project

We invested a total of **883 hours** on this project. Figure A.2 shows how time was distributed among the tasks that compose the project. We can clearly see that most of the time has been dedicated to the simulations. As we already explained in the previous section, the implementation of new features and correct modeling of the architecture took most of this time. Apart from that, time is more or less evenly distributed among the other tasks.

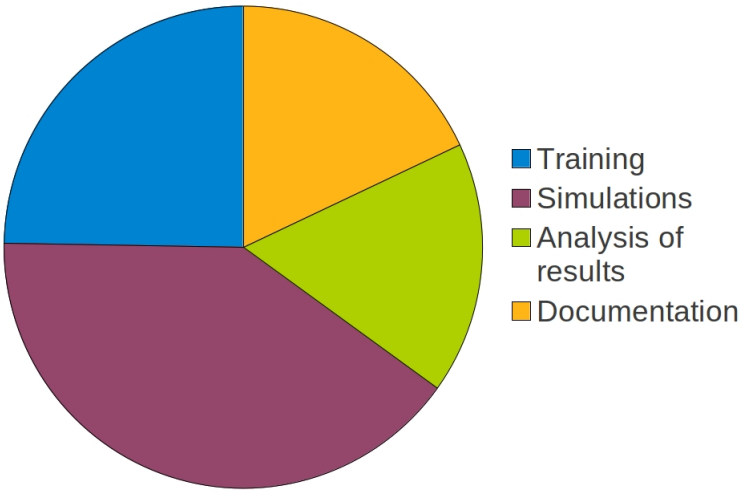


Figure A.2: Distribution of the time in the tasks that comprise the project.

Table A.1 details how many hours have been put into each one of the activities that compose the tasks.

A.3. Problems Faced

The biggest problems we had to face were related to the correct modeling of the architecture, specially the deadlock avoidance. Testing that it had been correctly implemented was very hard because it entailed creating a lot of congestion in the network and this caused other unrelated problems. Besides, when we thought that it was all working properly, we eventually found a particular case where a simulation failed only with certain parameters after several hours of execution. This errors took a long time to debug and delayed the development of the project.

Apart from that, we also had to face the fact that most of the simulations took several days to complete. This forced us to avoid leaving things to the last minute and to be extremely careful when launching new simulations because an error that involved repeating the simulations could

Table A.1: Number of hours invested on each task of the project.

Task	Number of hours
Training	218.5
Interconnection networks	134.5
Power and area	84.0
Simulations	353.0
Implementation and tests	287.5
Designing the experiments	65.5
Recopilation and analysis of results	151.0
Documentation	160.5
TOTAL NUMBER OF HOURS	**883**

easily entail a delay of a week.

Appendix B
CMP Architecture and Memory Subsystem

In this appendix we describe the system we will be modeling including a general description of the architecture and detailed description of the memory subsystem. A complete characterization of the interconnection network can be found in Section 3.2.

B.1. General description

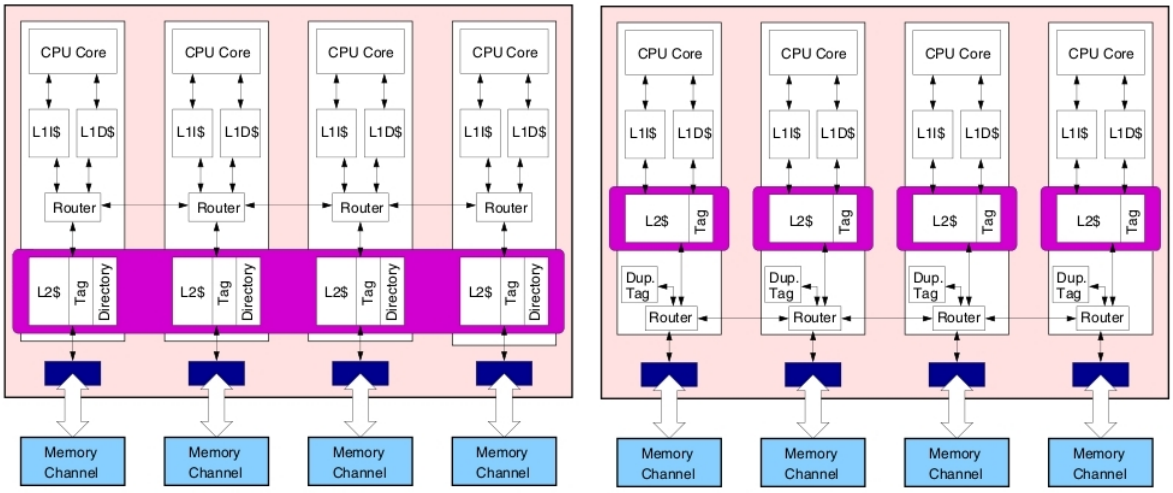
This study focuses on homogeneous chip multiprocessors (CMPs). The system is composed of several *tiles* connected by an interconnection network. On each tile we have a core with a private first level cache (L1) split into data and instructions and a bank of the shared second level cache (L2). On some of the tiles the is also a memory controller that connects the cores with the off-chip memory. We have a total of four memory controllers, uniformly distributed one on each side at the edge of the chip. Also, on each tile there is a router to connect both the L1 and the L2 to the network. For now, we are only studying configurations with a concentration degree of one, which means only one router is connected to each tile. Figure 3.1 depicts the chip and a tile with memory controller and the connections between the elements in the tile and the router. This router has two inputs and two outputs to connect the tile to other neighbouring ones.

We are modeling a system with simple Ultrasparc III Plus single-thread in-order cores. They are running Solaris 10 operating system and execute one instruction at a time. We include systems with 16 and 64 cores implemented in 32nm technology. Table 3.1 summarizes the key parameters of our system.

Another option to connect the elements would be to connect the local L1 directly to the L2 bank in the same tile, and connect only the L2 bank to the router. This way, traffic between the local L1 and L2 bank will not pollute the interconnection network. The drawback is that we would always have to access the local L2 bank, which will probably not be the host for the cache line we need. Since we have a shared L2, we decided to stick with our initial organization.

To model the system we looked into other architectures with similar characteristics, both from academia research papers and commercial processors. Zhang and Asanovič work with the configurations represented in Figure B.1, which are conceived to execute multithreaded workloads [49]. In their study, they present an architecture equivalent to ours with only four cores which has a shared L2 cache. In this configuration, the router is placed between the L1 and the L2, just like in our design. They also experiment with private L2 caches, in which case they use the

alternative we mentioned earlier to connect the elements to the routers. This option is better suited for architectures with a private L2 caches.



(a) Shared L2 **(b)** Private L2

Figure B.1: Chip architectures proposed in [49] by Zhang and Asanovìc.

Seiculescu *et al.*, who are focusing on interconnection network research, also include separate ports in their routers for the L1 and L2 shared caches, which include the directory information [39].

The Piranha architecture proposed by Barroso *et al.* integrates eight simple cores and a shared L2 cache on the same chip and is targeted at parallel commercial workloads [4]. The private L1 and shared L2 caches on a die are connected by an *Intra-Chip Switch* which works as a crossbar. This would be equivalent to any or our topologies where both the L1 and L2 caches have access to the interconnect. This architecture also includes the possibility of connecting several chips together.

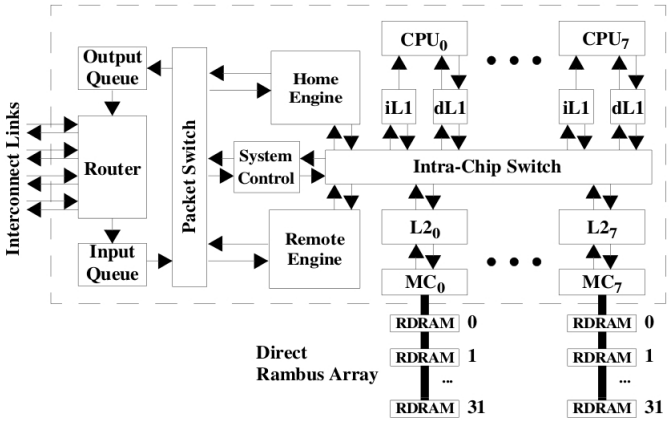


Figure B.2: Block diagram of a single-chip Piranha processing node [4].

The IBM POWER4 microarchitecture was designed for high-throughput multi-tasking environments [42]. On Figure B.3 we can see that the two processors on a chip share the three L2

banks through a crossbar called *Core Interface Unit (CIU)*.

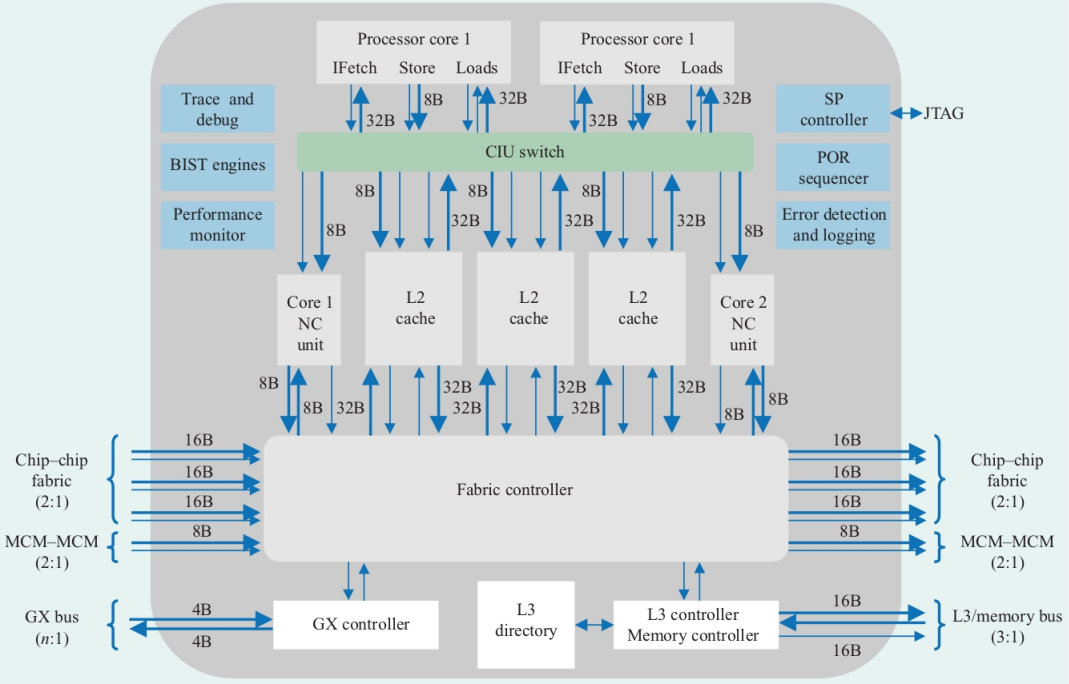


Figure B.3: POWER4 chip logical view [42].

Tilera's TILE*Pro*64 [43] is a multicore processor with 64 cores connected by an 8x8 mesh topology. It was to design for digital video processing, networking and cloud computing. Figure B.4 shows the components of a tile. We can see that both the L1 and L2 caches are connected to the router (which is called switch in the image).

In 2011, Intel proposed a multi-core processor architecture that integrates 48 cores in a single chip called *Single-chip Cloud Computer (SCC)* for academic and industrial research [20, 21]. Routers are connected in a 6x4 2D mesh, with two cores sharing each router. Cores communicate over the network utilizing a message passing architecture that allows data sharing with software maintained memory consistency. On each tile, the two L1 caches and L2 banks are connected to a *Mesh Interface Unit (MIU)* that sends the packets to the network router.

The Niagara2 architecture from Sun Microsystems has 8 SPARC cores, each supporting concurrent execution of 8 threads [36]. This processor was designed for servers, to help achieve a high throughput while maintaining low power consumption. The 8 cores are connected to the 8 banks of the L2 shared cache through a crossbar.

B.2. Memory coherence protocol

Chip multiprocessors are used to execute parallel applications (to reduce execution time) or independent programs on each core (to maximize throughput). In both cases, we need a memory coherence protocol. For parallel applications, it is used to maintain coherence among shared variables. For the independent programs, to allow migration of processes between cores. We are using a directory-based MESI coherence protocol. A MESI protocol has four states

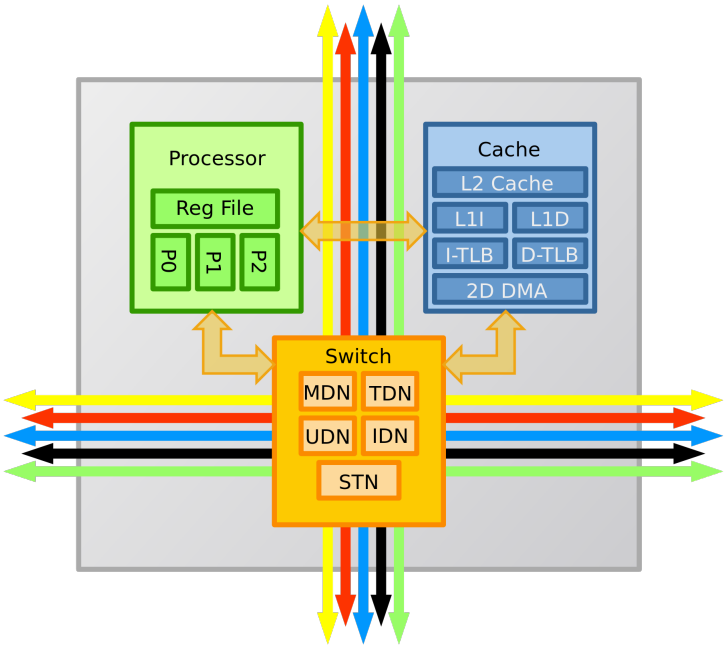


Figure B.4: Components of a tile of the TILEPro64 [43].

that depend on the coherence properties of the corresponding cache line: modified (M), exclusive (E), shared (S) and invalid (I). The L1 is a copy-back cache and uses write-allocate. Figures B.5 and B.6 present the coherence protocols for both the L1 and L2 caches. To facilitate the understanding we are omitting the intermediate states. These protocol is taken from the GEMS memory hierarchy simulator [33], where it is called MESI_SCMP_bankdirectory.

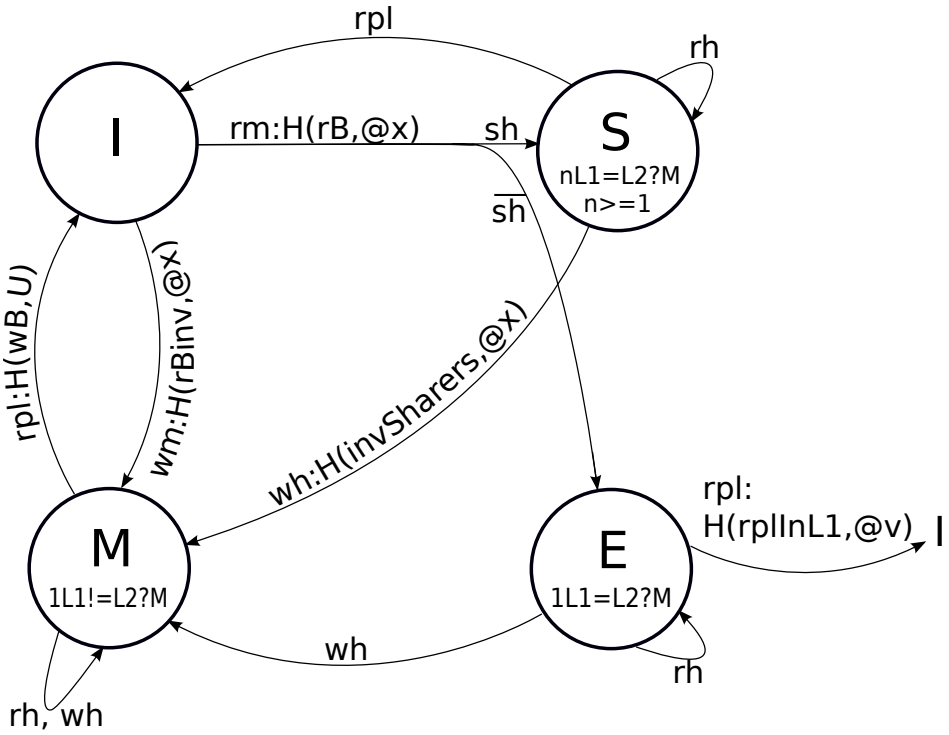
Inside the circles that represent the states, there is a short description of each of them indicating the state of the data contained in that cache line. In all of the states there is also a dirty bit indicating if data is consistent with main memory. The arrows represent the change of a cache line from one state to another. The labels in the arrows indicate the events and actions associated with the change of state and follow the pattern:

Event: Action1, Action2...

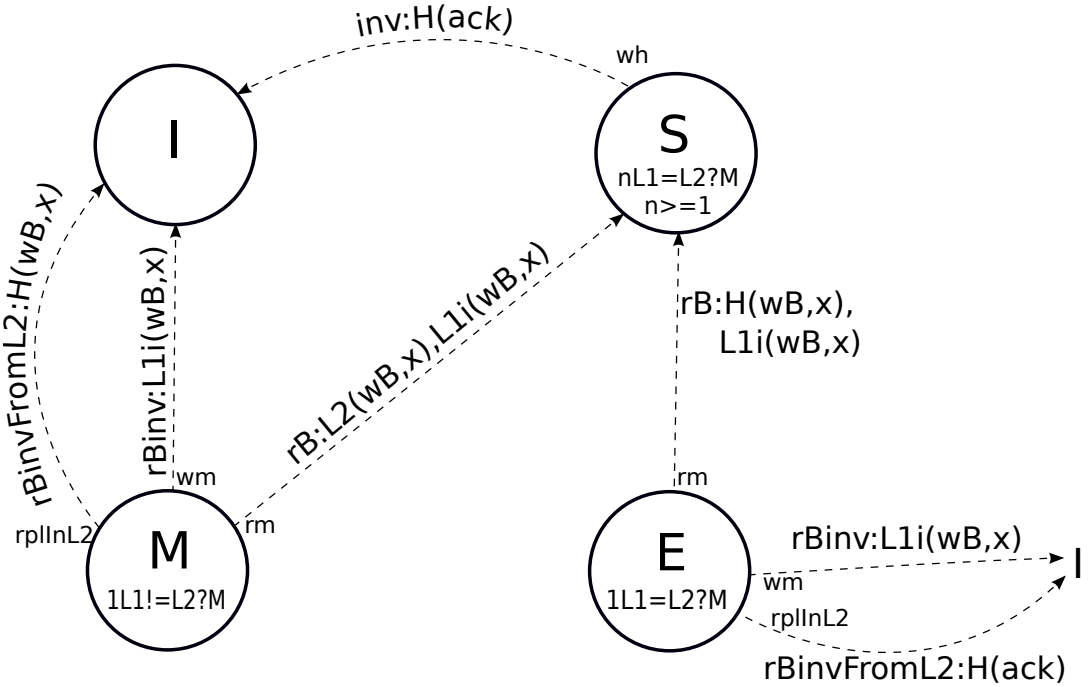
An event might not implicate any action, only a change of state. The actions usually indicate an event sent to another cache, with the following elements:

Destinatary(Event Sent, Parameters)

Table B.1 provides information about the nomenclature used in the diagrams. For the L1, arrows with continuous lines represent events coming from the processor. Arrows with dashed lines represent events coming from the L2, and include the original event that caused them in the base of the arrow. There is one case in the L1 diagram where an arrow gets divided into two lines that go to different states. This means that, depending on the sharing information received from the L2, data will end up on one state or the other. In some cases, the action is `L2+x`. This means that the L2 must update the data in a cache line it is already storing. Also, sometimes actions are separated by a forward slash instead of a comma. This means that the second action will execute after the result of the first action. In this cases, there are typically intermediate stages



(a) Events from the processor



(b) Events from the host L2 cache

Figure B.5: L1 cache coherence protocol.

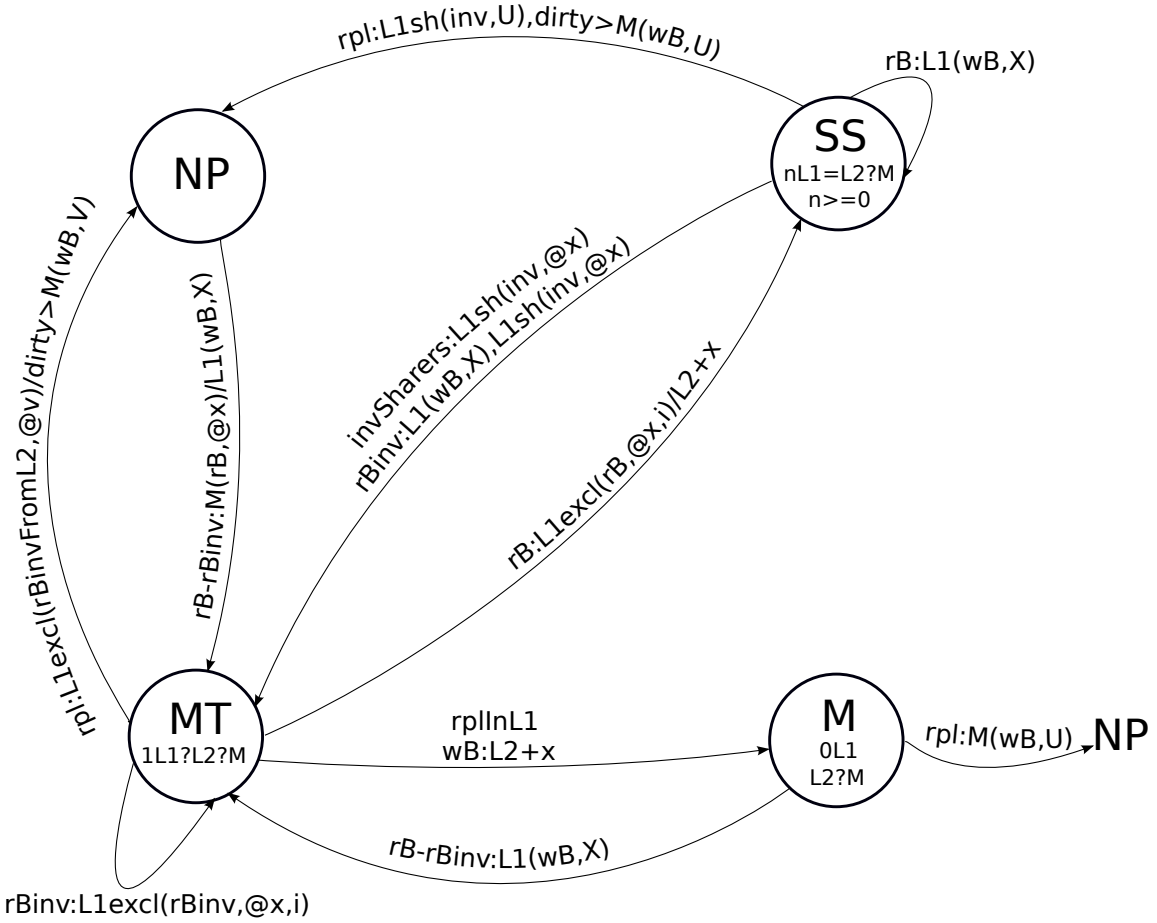


Figure B.6: L2 cache coherence protocol.

that have been omitted. Also, since there is also a dirty bit providing extra information, in some cases the execution of an action depends on the value of that bit. That has been indicated by `dirty>`, meaning that the ensuing action will only take place if the block is dirty. In the L2 we do not indicate updates of the sharers list in the directory.

About replacements in L1, there might be a notification sent to L2 or not. From the modified state, a notification is necessary because data is only up to date in the L1 and must be sent to the L2. From the exclusive state, the replacement is notified to the L2, but not data is sent because it not dirty. From the shared state, the replacement is silent. This means that the L2 is not notified and still thinks the L1 is storing the block. That is why the SS state in the L2 might mean that the block is not present in any L1 cache. When the L1 holds a block exclusively (either in the exclusive or the modified state), requests for that block are forwarded from the L2 host so that the L1 sends the data to the requestor and updates the state. When the L1 is in the exclusive state and gets a read block event from the L2, it means that another L1 has requested the block. So it sends the data to the requestor and moves it to a shared state. Here, the protocol sends the data to the L2 as well, which would not be needed since it is not dirty. This is because in the L2, data will be in state MT, where it does not know if data is consistent with the one in L1 or not and will always expect the data to update the cache line and move to the SS state. The sending of this data could be avoided but we have maintained it in the diagram to be consistent with the implementation.

The protocol includes a very basic implementation of the memory controller, which simply manages data requests and writebacks.

Table B.1: Nomenclature used in the cache coherence protocol diagrams.

L1 States	I	Invalid. The is no valid data in this cache line.
	S	Shared. Data is present in this L1 cache and might also be in other L1 caches. Data is consistent with the one stored in the L2, and might or might not be consistent with main memory.
	E	Exclusive. Data is only stored in this L1 cache and it is consistent with its L2 copy.
	M	Modified. Data is only stored in this L1 cache and it is not consistent with its L2 copy.
L2 States	NP	Not Present. The is no valid data in this cache line.
	SS	Shared. Data is present in 0, 1 or more L1 caches. Data in all of them is consistent with the one stored in the L2 but might or might not be consistent with main memory. It might be in 0 L1 caches because silent replacements in L1 are allowed.
	MT	Modified. Data is only stored in one L1 cache and data might or might not be consistent with the L2 copy. It also might or might not be consistent with main memory.
	M	Modified. Data is not stored in any L1. The L2 copy might be consistent or not with main memory.
Events	rh	Read hit
	rm	Read miss
	wh	Write hit
	wm	Write miss
	rpl	Replacement
	rplInL1	Replacement in the L1 cache
	rplInL2	Replacement in the L2 cache
	rB	Read block
	rBinv	Read block and invalidate
	rBinvFromL2	Read block and invalidate issued by the L2
	wB	Write block
	inv	Invalidate
	invSharers	Invalidate sharers
	ack	Acknowledgement

Destinataries	L1	L1 cache that performed the request
	L1sh	L1 caches that are sharers of the block
	L1excl	L1 cache that stores the block exclusively
	L1i	L1 cache that asked for the block
	H	Host, L2 cache where the block is or should be stored
	M	Main memory
Parameters	X	Data block
	@x	Data address
	U	Victim block, block issued for replacement
	@v	Address of the victim block
	i	This is included when the destinatary needs to know which L1 it will have to send the data to

Appendix C
Methodology and Experimental Environment

This appendix contains details about the metrics, the benchmarks and the simulation environment used to characterize the interconnection networks.

C.1. Metrics

We have chosen several metrics to characterize the behaviour of the interconnection network. Since we are interested in the impact the interconnect has in the whole system, we have also included metrics that reflect the overall performance and the behaviour of the memory hierarchy.

The metrics included in this study are the following:

- **Performance.** For the parallel workloads, we analyse the number of processor cycles it takes to complete the execution of the parallel section of the applications with all the configurations. With the multiprogrammed workloads we execute a fixed number of cycles, so we count the number of completed instructions.
- **Miss rate**. We study the miss rate of the applications in both levels of the memory hierarchy for both data and instructions and relate it to the demands placed on the interconnect.
- **Node throughput.** We check the number of flits injected and received by each node. We analyse the total number of flits and the average flits per cycle and node, so as to be able to compare between configurations with different tile count. We evaluate if all the nodes inject and even amount of packets and try to find hotspots which might be located in the access to the memory controllers.
- **Link utilization.** Similar to the analysis of the tile throughput, we represent the utilization of the links in flits per cycle and look for hotspots.
- **Number of hops.** If the average distance travelled by flits is close to the average distance of the network, we could state that the interconnect is being used evenly. A skewed average number of hops would indicate that there are more messages accessing nodes that are very close or very far.
- **Network latency.** By looking into the delay introduced by the network we can find out if it is a critical part of the system that dominates performance results. We express this latency in network cycles (which in our case, is equivalent to processor cycles).

- **Average virtual channel load.** We examine the average occupancy of the virtual channels in flits per cycle. The ideal situation is to have a balanced load across all virtual channels.
- **Arbitration requests.** We check the number of requests each flit has to make for virtual channel allocation and switch allocation until the request is granted. This gives us an idea of the contention in the network.
- **Energy.** We measure the energy expended by the interconnect (the links and the stages of the routers) and by the caches. As future work, we intend to compare this values to the power needed by the processor so as to get a more general view of the whole system.
- **Area.** We compare the area needed by the different topologies and by the memory hierarchy.

C.2. Workloads

The chip multiprocessors we are focusing on may be used to execute parallel applications in order to reduce execution time or for multiprogrammed workloads (execution of independent programs on each core), to increase throughput. When choosing which applications to include in our experiments, we are interested in the behaviour they will have on the interconnection network which depends on characteristics such as communication and traffic. We are using a shared-memory multicore system, so when talking about communication we refer to the memory accesses that are used to exchange information between cores (writing a value that will be read by another core or reading a value previously written by another core), that is, true sharing accesses [5]. Communication accesses always involve a cache miss, but the evaluation of the parameters has been done independently. So a benchmark could have a lot of communication compared to the others, but still have a lower miss rate. The same thing applies to L1 and L2 misses, we may have a high L2 miss rate with a low L1 miss rate. Traffic is a more general term that represents movement of data to and from the caches, so it includes all the packets injected into the network. Since data is uniformly distributed across the L2 banks, memory accesses might need to use the network without entailing communication with other cores. Most of the traffic is due to L1 misses, which are more frequent than L2 misses which involve access to the memory controllers.

We are using a selection of shared-memory parallel applications from PARSEC and SPLASH2 and a multiprogrammed workload made up of SPEC CPU2006 benchmarks. The performance of these applications may be limited by the interconnection network and we intend to establish to what extent this is actually happening. We would like to work with pressing applications that offered a very high miss rate, a large amount of communication and traffic and good scaling. To be fair, we tried to put together a set of applications with varying characteristics in terms of the axes previously mentioned. In the following sections, we go into the specifics of the selection of benchmarks for the three benchmark suites considered.

PARSEC

PARSEC is a benchmark suite composed of multithreaded programs implemented with OpenMP and pthreads [11, 9, 10, 5, 7, 8]. It focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. We performed a complete study of this benchmark suite and presented our results in a poster session at the HiPEAC international conference in January 2012 [32].

In Table C.1 we include all benchmarks from the PARSEC benchmark suite with a short description of each of them and the characteristics that were taken into account to choose which applications to use in our study. The benchmarks chosen are `blackscholes`, `canneal`, `fluidanimate`, `swaptions` and `x264` (they are shown with bold font in the table). They have all been executed with the large input except for `x264`, for which we have used the medium input due the large simulation time. `Canneal` and `Swaptions` have been chosen for their large amount of traffic; `x264` has been included because of the large amount of communication between cores and `blackscholes` and `fluidanimate` are some of the benchmarks with better scaling properties. We simulate the whole parallel region of the applications. To simplify the understanding of the information, for most characteristics we simply describe their values verbally. This is enough to compare the benchmarks with each other.

We got miss rates and traffic from [11, 47], scaling information from [7] and sharing data from [5]. For `raytrace`, we don't have much information because it was included in the second version of the benchmark suite, after the papers we are referencing came out.

Table C.1: Characteristics of the PARSEC benchmark suite applications. Chosen applications appear in boldface.

Benchmark	Domain	L1 miss rate	L2 miss rate	Scaling	Comm	Cache Traffic
Blackscholes	Financial Analysis	Medium	Low	Good	Low	Medium
Bodytrack	Computer Vision	Medium	Medium	Poor	Medium	Medium
Canneal	Engineering	High	High	Poor	High	High
Dedup	Enterprise Storage	Low	Medium	Poor	High	Low
Facesim	Animation	Medium	Medium	Good	High	Medium
Ferret	Similarity Search	Medium	Medium	Average	Medium	Medium
Fluidanimate	Animation	Medium	Low	Good	Medium	Medium
Freqmine	Data Mining	Medium	Low	Average	High	Medium
Raytrace	Rendering	Low	Low	-	-	-
Streamcluster	Data Mining	High	High	Average	Medium	High
Swaptions	Financial Analysis	High	Low	Good	Low	High
Vips	Media Processing	Medium	Medium	Good	Medium	Medium
X264	Media Processing	Medium	Medium	Average	High	Medium

More information on number of instructions, writes, reads, barriers and locks can be found in [11].

SPLASH2

SPLASH2 is a mature benchmark suite containing a variety of high performance computing and graphics applications [46, 40, 10, 5]. Shared-memory parallelism is explicitly written by using PARMACS, a library of parallel macros that allow an architecture-independent implementation [29, 30].

As we did in the previous section, in Table C.2 we include all the applications and a brief compilation of its characteristics. The chosen benchmarks are `barnes`, `fmm`, `ocean`, `radiosity`, `volrend` and `water-spatial`. `Ocean` is the application with higher miss rates and traffic. `Water-spatial` and `volrend` have good scaling. The latter also has a high L2 miss rate, which means it will need to access off-chip memory frequently. `Barnes`, `Fmm` and `Radiosity` have been included as examples of applications that will not pressure the interconnection network. As we do with PARSEC, we simulate the parallel region of the applications. All the information has been collected from [46] and [5].

Table C.2: Characteristics of the SPLASH2 benchmark suite applications. Chosen applications appear in boldface.

Benchmark	Domain	L1 Miss rate	L2 Miss rate	Scaling	Comm	Cache Traffic
`Barnes`	n-body Simulation	Medium	Low	Good	Medium	Medium
`Cholesky`	Matrix Factorization	High	High	Poor	Medium	High
`Fft`	Complex 1-D FFT	High	High	Good	High	High
`Fmm`	Fast Multipole n-body	Medium	Medium	Average	Medium	Medium
`Lu`	Matrix Triangulation	Medium	Low	Poor	High	Medium
`Ocean`	Ocean Current Simul.	High	Medium	Good	High	High
`Radiosity`	Graphics	Low	Low	Poor	Low	Low
`Radix`	Integer Sort	Medium	High	Poor	High	Medium
`Raytrace`	3D Rendering	Low	High	Average	Medium	Low
`Volrend`	3D Rendering	Low	High	Good	Low	Low
`Water-nsq.`	Molecular Dynamics	Medium	Medium	Good	Medium	Medium
`Water-spatial`	Molecular Dynamics	Medium	Medium	Good	Low	Medium

More information on number of instructions, writes, reads, barriers and locks can be found in [46].

SPEC CPU2006

SPEC CPU2006 is a benchmark suite composed of single threaded applications written in C, C++ and Fortran [41]. We have used it to build a multiprogrammed workload in which we run one application on each core, binding applications to cores so that no migration occurs. The threads that are being executed in each core are independent, so the only traffic in the network will be caused by cache misses and replacements. Since no data will be shared and no migration is allowed, there will be no additional coherency messages.

In this case we chose the applications with the highest footprint and working set size (according to [18, 16]) so as to determine the potential bottlenecks due to the interconnect. The footprint is the amount of memory a program uses when it is executed. We will represent it with the virtual size, which is the total address space the operating system has reserved for the application. That metric is an appropriate measure of the amount of memory that an application occupies, but do not tell us how much memory an application actually uses. The working set is an estimate of how much memory is being actively used by an application. A larger working set implies higher miss rates in the memory hierarchy, and therefore, more data requests and replies.

The 16 chosen applications are listed and briefly described in the Table C.3.

Table C.3: Characteristics of the SPEC CPU2006 applications used.

Benchmark	Domain	Virtual size (MB)	Working set size (MB)
`400.perlbench`	PERL Programming Language	581	51.3
`401.bzip2`	Compression	856	24.4
`403.gcc`	C Compiler	933	70.7
`429.mcf`	Combinatorial Optimization	845	680.8
`458.sjeng`	Artificial Intelligence: Chess	180	57.7
`462.libquantum`	Physics: Quantum Computing	105	32.7
`410.bwaves`	Fluid Dynamics	894	474.3
`433.milc`	Physics: Quantum Chromodynamics	677	230.8
`434.zeusmp`	Physics/CFD	1138	270.1
`437.leslie3d`	Fluid Dynamics	142	75.2
`447.dealII`	Finite Element Analysis	566	14.7
`450.soplex`	Linear Programming, Optimization	626	201.5
`450.GemsFDTD`	Computational Electromagnetics	850	800.0
`450.lmb`	Fluid Dynamics	417	402.0
`450.wrf`	Water Prediction	718	163.5
`450.sphinx3`	Speech Recognition	49	10.6

To build the workload for the 16-core architectures we have executed each application once, binding each one of them to a different core. For the 64-core architectures we have used the applications four times assigning them to the cores consecutively (this should have better been done randomly, that is left for future work). To execute this workload, we first warm up the

caches for 200 million cycles and then execute for 500 million cycles.

C.3. Simulation Environment

We have obtained our results using simulation, which is a common resource in computer architecture for the exploration of the design space of new computing systems. We have chosen a set of tools used extensively in this field. Wind River Simics (usually called just Simics) allows us to simulate the full-system and can be configured to model multiprocessors, embedded systems, telecom switches, clusters and networks of all those items [31]. It can directly execute operating systems and simulate real applications offering precise results. It is a commercial simulator and it is not open source. Simics is usually used in conjunction with GEMS (General Execution-Driven Multiprocessor Simulator), created in the University of Wisconsin [33]. It provides modules for the study of the performance of the memory system and microprocessors. GEMS comprises Ruby, which simulates caches, the coherence protocol and the interconnection network, and Opal, for out of order execution. Simics behaves as a functional simulator, that is, it is simply an interpreter that executes the instructions. It communicates with the Ruby module from GEMS, which will manage memory accesses and provide timing information. GARNET is a detailed interconnection network simulator integrated in Ruby that was implemented in Princeton University [1]. Figure C.1 illustrates the previous description. The random tester and the microbenchmark modules can also serve as a source of memory operation requests for Ruby, but we are not using them. We are not using Opal either, since we are simulating simple in-order processors.

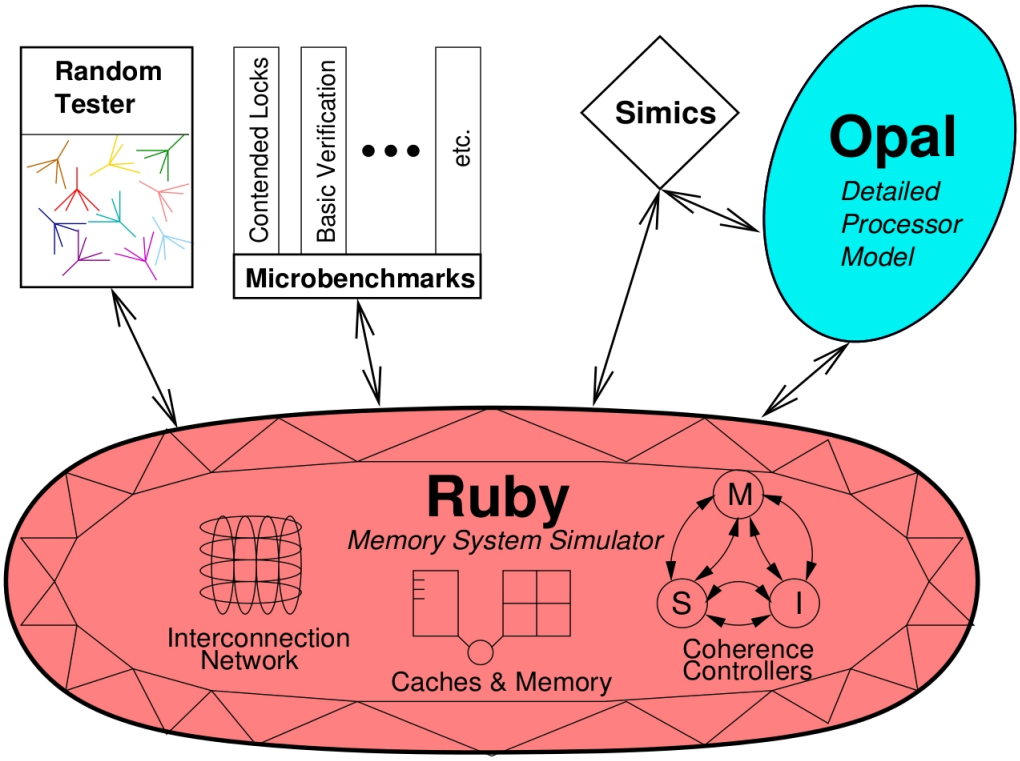


Figure C.1: View of the GEMS architecture.

We are carefully modeling all the components of the system and performing full-system simulation with simple single-threaded cores and directory-based coherence. We have had to

include several modifications to the Ruby module to produce the statistics and support the topologies we wanted to study. For the power analysis we used two additional tools, CACTI and Orion. We describe this in detail in the following sections.

New Functionalities Incorporated into the Simulator

We are first going to review the modifications needed to simulate the system with our desired features.

We are interested in simulating three topologies: mesh, torus and ring. Garnet implements several topologies and gives the opportunity to define any others by using a *network file*. In this file we simply need to indicate which elements to include in each node (processor and L1 cache, L2 cache and memory controller) and how to connect them. To model the mesh topology, we use these configuration files. The torus is one of the topologies already implemented in Garnet, but we noticed that it was only built properly if each processor was placed in a different chip, which was not what we needed. Besides, when using a torus we introduce loops and we need to deal with the deadlocks that might occur (see the topology description in Chapter 3). The torus provided by Garnet does not take this into account, so we decided to construct the torus and ring topologies with a network file and implement the deadlock avoidance.

Originally, Garnet always performed virtual channel allocation using a round-robin system. To avoid deadlocks caused by loops in the torus and ring topologies, we implemented the naive and balanced deadlock avoidance methods already described in Chapter 3.

When the torus and ring topologies are built from the network files, the routing table storing the minimal paths between all nodes is created. If there are two minimal paths between two nodes (this happens in every loop with an even number of nodes), the same one is always selected in detriment of the other depending on which link was written first in the network file. This causes some links to be used more frequently than others. To avoid this, we statically force the minimal path to be only one of the two, alternating the direction the messages will follow depending on whether the source node identifier is even or odd [38].

Additionally, we wanted our routers to have four virtual channels per input. As we explained in Chapter 3, we need two virtual channels for deadlock avoidance and two virtual networks to separate request and reply messages. In GEMS, the virtual networks are defined in the cache coherence protocol and each message is assigned to one virtual network depending on its type (request from L1 to L2, request from L2 to the memory controller,...). So we modified the protocol to assign all request messages to one virtual network and all reply messages to the other one.

We also had to solve the deadlock caused by coherence messages overtaking other previous ones targeting the same cache line, already explained in Chapter 3. We consider acknowledgements as high priority messages, because they mean there is cache line being blocked waiting for them. The only thing the original protocol did to deal with this was to assign a single virtual network to messages that should be prioritized and arrive sooner to their destination. This way, those messages were more independent from messages with different types and were expected to always arrive sooner. We implemented another version of the protocol with three virtual networks, one exclusively dedicated to these high priority messages. This solved the problem, but forced us to have routers with six virtual channels, which was not our desired configuration. We also implemented a partial ordering in the interconnection network. Whenever a message is at a

router, we check if there is another message with higher priority waiting to get to same cache line. In that case, the first message will never be granted the virtual channel or the access to the switch it needs to continue. This also solved the problem and allowed us to keep only four virtual channels.

We are working with the *fixed pipeline* offered by Garnet, which means every hop in our network takes five cycles (four in the router and one in the link, see Figure 3.4). We wanted to have the possibility of experimenting with routers with a lower latency, as explained in Chapter 3. Garnet also includes a *flexible pipeline* model, but everything we had already implemented was working only for the pipeline with a fixed number of stages. To solve this, we included the option of reconfiguring the router pipeline by combining consecutive stages so that they were executed on the same network cycle. This allows us end up with a one-cycle router. To implement this, we had to include the possibility of scheduling an event (the wake up of the next stage) for the current cycle, not only for a future cycle. This minor modification usually has side effects which cause further problems difficult to detect. For example, we noticed that we were sometimes scheduling the waking up of a stage for the current cycle but it had already woken up. Since a stage can't be executed twice in the same cycle, that stage never got executed, leading to a deadlock. The general problem was that all the events scheduled for a particular cycle were serviced in no particular order. To avoid this we included an ordering of the events in the pipeline so that they were executed backwards except for those stages that had to be executed in the same cycle.

We also needed to implement the computation of new statistics. Garnet already offers average link utilization, virtual channel load and network latency.

We have implemented a count of injected and received messages per network interface (each L1, L2 block and memory controller). This way, we can see how many messages and flits each node has injected and received. We also get this count per message type.

We print more detailed information about the utilization of each link. This will allow us to detect hotspots in the network.

We have included the count of occurrences for message latency and hop count. Later, we will be able to use that information to get the average, variance or quartiles to represent the data as we see fit.

The count of accesses and misses in the L1 and L2 originally implemented in Ruby was not working properly. We fixed it and reduced the granularity to get the information divided into message types. This accurate information was later used to calculate the energy expended by the memory hierarchy. We also included a count of messages sent by caches and memory controllers broken into coherence types.

To check if the network is overloaded we have included a count of the number of times virtual channel and switch allocation are requested but arbitration fails.

Power and Area Analysis

To get the area and the energy expended by the network we used a circuit modeling tool called Orion 2.0 [23, 24]. The version of Orion included in Garnet was 1.0, which didn't give

accurate results compared to the new version. The first thing we did was to integrate the new version in our Ruby module. Orion 2.0 gives us information about the area, leakage and dynamic power for the links, the router stages (input, virtual channel allocation, switch allocation and crossbar traversal) and the propagation of the clock.

For the area and energy expended by the memory hierarchy we used CACTI 6.5, which is another circuit modeling tool for estimating access time, cycle time, area, leakage, and dynamic power [35]. We get into further detail about the estimation of the energy expended by the caches in the next section.

The technology files in both tools have been matched so as to be able to compare the results obtain from the two models. Some parameters have been taken from the work of Gracia *et al.* [17] and accuracy has been further improved by approximating our values to the ones used in INTEL architectures [22].

C.4. Estimating the Energy Expended by Caches

In this section we explain how to calculate the energy expended by the L1 and L2 caches with a MESI protocol (in particular, the MESI_SCMP protocol on GEMS) using the information obtained from CACTI 6.5 [35]. It can help us to better understand the performance tradeoffs inherent in memory system organizations.

To calculate power we may apply the following formula

$$Power = DynamicPower + LeakagePower$$
$$Power = C * V_{dd}^2 * f * \alpha + I_{leak} * V_{dd}$$

where C is the capacitance, V_{dd} is the supply voltage, f is the frequency, α is the activity factor and I_{leak} is the leakage current. To get the energy expended we only need to multiply the power by the time our system has been running.

Alternatively, we can compute the dynamic energy by counting all the switching events and multiplying them by their respective energy consumptions. This is the way we proceeded in our power analysis.

Static Energy

CACTI gives us the leakage of the caches, which we consider to be the same as the static energy since the contribution of its other components is negligible. To get the total energy expended, we need to multiply that by the total time, which is the number of cycles the caches were used divided by the processor frequency.

Dynamic Energy

CACTI tells us how much energy the cache consumes when performing a switching event, such as a tag read (TR), tag write (TW), data read (DR) and data write (DW). We then have to count the events that take place in the cache when we run the program (read hit, read miss, replacement...) and multiply them by their consumption. In our configuration, the tag array contains extra bits for storing the cache line state and the valid, dirty and pseudo-LRU bits.

In a cache, the tag and data arrays can be accessed sequentially or in parallel. Usually, the L1 supports parallel access to allow a faster response, and the L2 is accessed sequentially. Independently of the access mode, writes must be performed sequentially checking the tag first and writing the data afterwards if we have experienced a hit. For the write to be performed in one cycle, we can store the data in a write buffer and write in the cache later.

As an example, we are going to explain the operations performed when accessing a parallel cache for a couple of events.

- Read miss (rm). We perform a first access in which we notice the line is not in the cache, so we have a miss. In a second access, after receiving the line from wherever it is, we do the data refill. We have two options depending on when we decide which line we are going to evict: eager victim selection (we choose which line to replace during the first access) and lazy victim selection (in the first access we just realize the block is not in the cache, so we have to choose the victim in the second access). We also have to remember to add the energy expended when replacing the victim block in case it is dirty. For each of the two options, the operations performed are the following:
  - Eager victim selection. In the first access we will have TR+DR. The DR is not needed because in this access we discover that the line is not in the cache, but it is always performed in parallel. In that first access we have also chosen what the victim line will be and invalidated it. In the second access we will just have to insert the new line (TW+DW). This is the method used in our cache configurations.
  - Lazy victim selection. We start again by performing TR+DR. We notice it is a misfs, and forward the request, but the cache line is remains available until we receive the data. Now, in the second access we need to start with a reading access again to choose the victim so the operations would be (TR+DR)+(TW+DW).
- Write hit (wh). In this case, the operations will be TR+DW+TW*0.2. The tag write included corresponds to the update of the dirty bit and state when the line was in a shared or exclusive state. We assume that every time we modify some of the extra bits contained in the tag array, it will consume a 20% of the total energy for a tag write, since we are not rewriting all the bits. If we were in the modified state, that tag write would not be needed.

For simplicity, we will not distinguish the reading or writing of a whole cache line from the reading or writing of a single byte or word.

L1 cache. Parallel Access

In Table C.4 we explain the operations performed in the L1 cache for each event and directly connect them to the states and event names used by GEMS for the coherence protocol. We assume the L1 will have a parallel access to the tag and data. Table C.6 contains the meaning of the acronyms used in the power analysis.

L2 cache. Sequential Access

In Table C.5 we explain the operations performed in the L2 cache for each event and directly connect them to the states and event names used by GEMS for the coherence protocol. In this case, we assume the L2 will have a sequential access to the tag and data. Again, the acronyms used can be found in Table C.6.

Table C.4: Operations performed in the L1 cache for each event

Event	Protocol event	Current State	Operations	Comments
rh	Load, Ifetch	S, E, M	TR+DR	
rm	Load, Ifetch	NP, I	TR+DR+TW+DW	We read tag and data in parallel before knowing it's a miss. We assume eager victim selection.
wh	Store	S, E	TR+DW+TW*0.2	We update only the state and the dirty bit.
wh	Store	M	TR+DW	We are already in a modified state, so no need to update the tag.
wm	Store	NP, I	TR+TW+DW	
rpl	L1Repl	S, E	TR+TW*0.2	Look for the line and update the state. Since it is clean, there's no need to copy it.
rpl	L1Repl	M	TR+TW*0.2+DR	In this case, the block has been modified so we need to read the data (writeback to L2).
inv	inv	S, E	TR+TW*0.2	We need to find the block and update the state.
inv	inv	M	TR+TW*0.2+DR	We might be the ones who have the block updated, so we need to send it.
req from L2	Fwd GETS, GETX, GETINSTR	E, M	TR+TW*0.2+DR	We need to find the block, update the state and send the data.

Table C.5: Operations performed in the L2 cache for each event

Event	Protocol event	Current State	Operations	Comments
rh	GETS, GETINSTR	SS	TR+DR+TW*0.1	Send the block to another L1. We also update the sharers in the tag.
rh	GETS, GETINSTR	M	TR+DR+TW*0.1	Send the block to another L1 and update the state (in the M state no L1 can have the block) and sharers.
rh	GETS, GETINSTR	MT	TR+TW*0.1	We have the block but it might be clean only in the L1, so we forward the request to the owner (L1). Update the sharers.
rh	GETX	SS, M	TR+DR+TW*0.1	Another L1 wants to read the block and write on it, so we send the block and change the state and sharers.
rh	GETX	MT	TR+TW*0.1	Another L1 wants to read the block and write on it, but an L1 has it exclusively. We just forward the request to the owner and modify the sharers.
rm	GETS, GETINSTR, GETX	NP	TR+TW+DW	
wh	L1upgrade	SS	TR+TW*0.1	The block just needs to change its state to be exclusive.
wb	PUTX	MT	TR+TW*0.1+DW	This is a write back. We change the state and copy the data.
rpl	L2rpl	SS	TR+TW*0.1+DR	The block is dirty so I have to write back to memory.
rpl	L2rpl clean	SS	TR+TW*0.1	No writeback needed.
rpl	L2rpl	M	TR+TW*0.1+DR	
rpl	L2rpl clean	M	TR+TW*0.1	
rpl	L2rpl, L2rpl clean	MT	TR+TW*0.1	I forward the request to the owner (L1), so I do not have to read the data here.

Table C.6: Meaning of the acronyms used for power analysis.

Acronym	**Meaning**
TR, TW	Tag read, tag write
DR, DW	Data read, data write
rh, rm	Read hit, read miss
wh, wm	Write hit, write miss
rpl	Replacement
inv	Invalidation
req from L2	Request forwarded from L2
wb	Writeback
Ifetch	Instruction fetch
L1Repl	Replacement in the L1 hline
GETS	Get shared data. It is a data request for a read access
GETX	Get exclusive data. It is a data request for a write access
GETINSTR	Get instruction
Fwd GETS, GETX, GETINSTR	Forwarded requests from L2 to L1
L1upgrade	Upgrade data from being shared to being exclusive due a write access
PUTX	Writeback
L2repl	Replacement in the L2
L2repl clean	Replacement in the L2 but no data sent because it has not been modified
S, E, M, NP, I	L1 states: Shared, Exclusive, Modified, Not Present and Invalid
SS, M, MT, NP	L2 states: Shared (Data is present in 0 or more L1 caches and has not been modified), Modified (Data is only in the L2 and might have been modified), Modified (Data is in one L1 and might have been modified), Not Present.

Appendix D
Results

This appendix contains the results of our analysis for 16 and 64-core architectures, simulating both parallel and multiprogrammed workloads. We focus on the comparison of performance, traffic distribution, power and area for the mesh, torus and ring topologies. We include a ring with increased bandwidth and one with 3-cycle routers. We conclude by indicating which topology should be used for the 16 and 64-core systems.

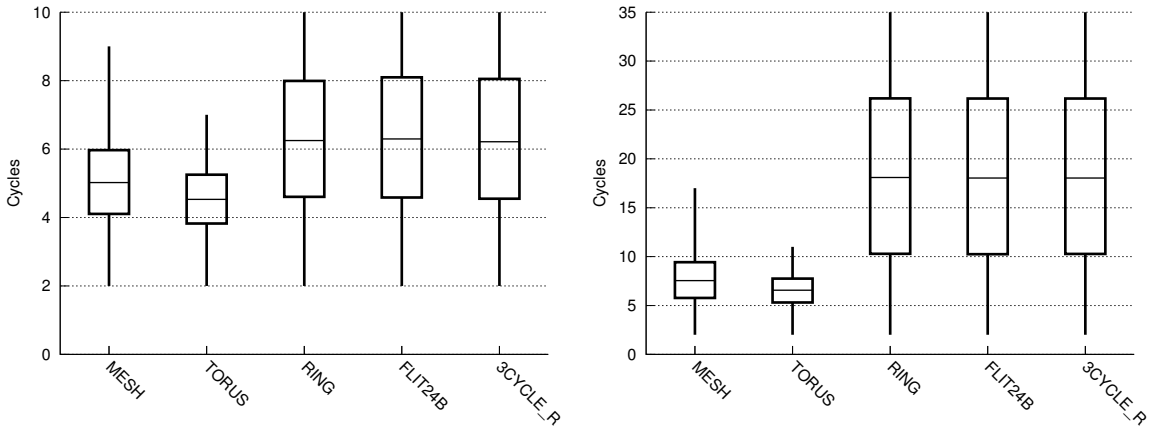
D.1. Performance

Our objective is to compare the behaviour of the ring, mesh and torus topologies. The ring a simple topology that is expected to have worse performance but less power and area costs. The torus is the most complex topology out of all three. It has more links that provide lower communication latency but suffers from higher energy consumption. The mesh is an intermediate option used very frequently due to its regularity. To benefit from the small area of the original ring topology, two additional rings have been included: one with higher bandwidth where flits are 24 bytes long (instead of the original 16 bytes), and one with 3-cycle routers (instead of 4-cycle routers).

To compare the impact of the network configurations on performance, we are studying the number of processor cycles it takes for the parallel workloads to complete the parallel section. For the multiprogrammed workloads, we check how many instructions get executed in 500 million cycles. Figure 5.1 shows these values normalized to the mesh topology. We can see that we achieve the best performance with the torus topology, closely followed by the mesh. In 16-core architectures differences between topologies are much smaller, with the ring with 3-cycle routers being very similar to the mesh for some applications. In 64-core applications, the performance of the ring topologies drops significantly. The ring with higher bandwidth (with 24B flits and links, as opposed to the standard 16B) performs only slightly better than the original ring, which does not compensate for the increase in area and power. The ring with 3-cycle routers instead of 4-cycle ones also performs better than the original ring, but is still much worse than the mesh and torus.

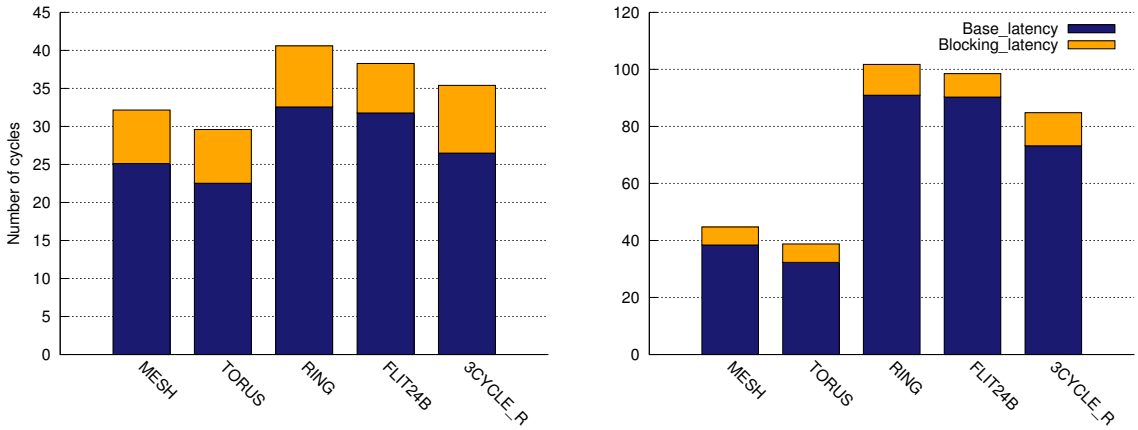
The differences in performance are a direct consequence of the number of hops it takes a message to go from its source to its destination. In Figure D.1 we present candle sticks for the aggregate hop count for the 16 and the 64-core architectures. Intuitively, we could consider that we are executing all the applications one after the other on each system configuration. The differences among the workloads are extremely small. Both the median and the variability of the hop count are much bigger for the ring topologies, especially with 64 cores, which is where we

saw more pronounced performance drops. Hop count for the ring topologies is the same in all cases. The difference in performance for the ring with 24B-flits is because data messages will be only three flits long instead of five, which will reduce the serialization latency.



(a) 16 cores **(b)** 64 cores

Figure D.1: Average hop count for 16 and 64 cores. All applications have been considered together, as if they were executed back to back. There are no differences between parallel and multipgorammed workloads, so results are combined. We present candlesticks, where we can see the minium and maximum values and the three quartiles. Note that scales are different



(a) 16 cores **(b)** 64 cores

Figure D.2: Average network latency in number of cycles broken down into base and blocking latency for 16 and 64 cores. All applications have been considered together, as if they were executed one after the other. There are no differences between parallel and multipgorammed workloads, so results are presented combined.

Performance could also drop because congestion on the network is holding messages on the network. However, that is not the case. We have checked that, although it takes long to traverse the network, resources are idle most of the time. Figure D.3 shows average link utilization in flits per cycle. We can see that each link is being used, in average, no more 10 times every 100 cycles, which means the network is not being highly used. Figure D.2 represents the network latency split in the base latency (cycles it would take packets to traverse the network without contention) and blocking latency (extra time due to contention). We can clearly see that the

blocking latency is a small percentage of the total latency. We also detect the differences in latency that were mentioned in the previous paragraph and are responsible for the performance variances between the ring topologies. Figure D.4 depicts the number of times each flit had to request virtual channel allocation and switch allocation until it was granted. Virtual channel allocation is only performed by the head flit of each message. The ideal situation would be have one request per flit, meaning they got the resources in the first try. The graph shows that flits need to request allocation always less than 1.2 times, which means requests are frequently granted. There are no significant differences between topologies. The ring topologies have slightly more non-granted requests because, since there are less links and less virtual channels, the same amount of traffic needs to compete for less resources. The torus has more arbitration failures than the mesh because it condenses the same amount of traffic in a shorter execution time. VC allocation fails more frequently than SW allocation, which is logical since some flits are turned back at VC allocation and, therefore, there are less flits contending for the crossbar. Again, this points out the low congestion of the network. There still could be congestion burstiness that is not showing up because we are considering temporal averages of the metrics. It is left for future work to check if there are localized congestion bursts in certain moments of the simulations.

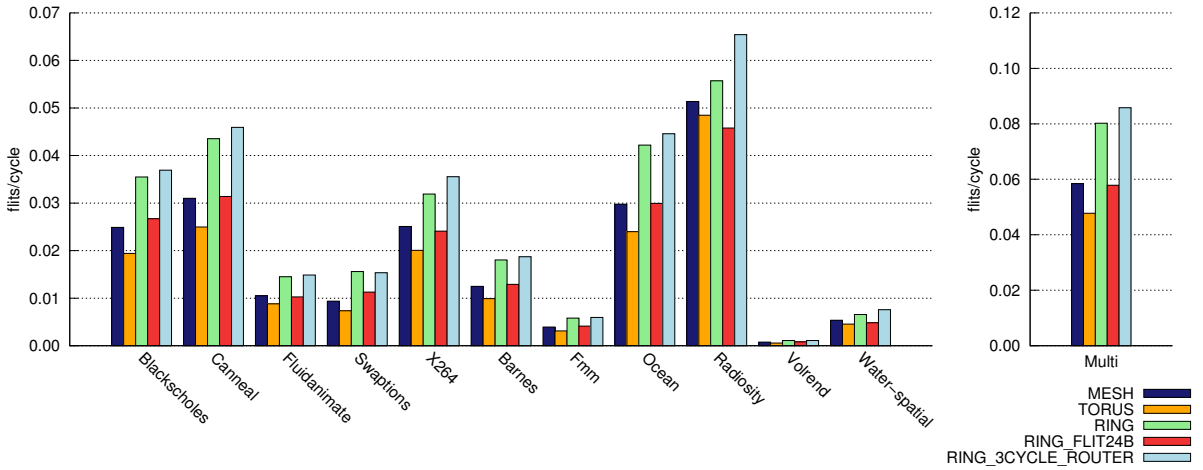
These results ratify the conclusions of Sanchez *et al.*, which point out that the number of hops is the most critical parameter of the network [37].

D.2. Non uniform traffic distribution

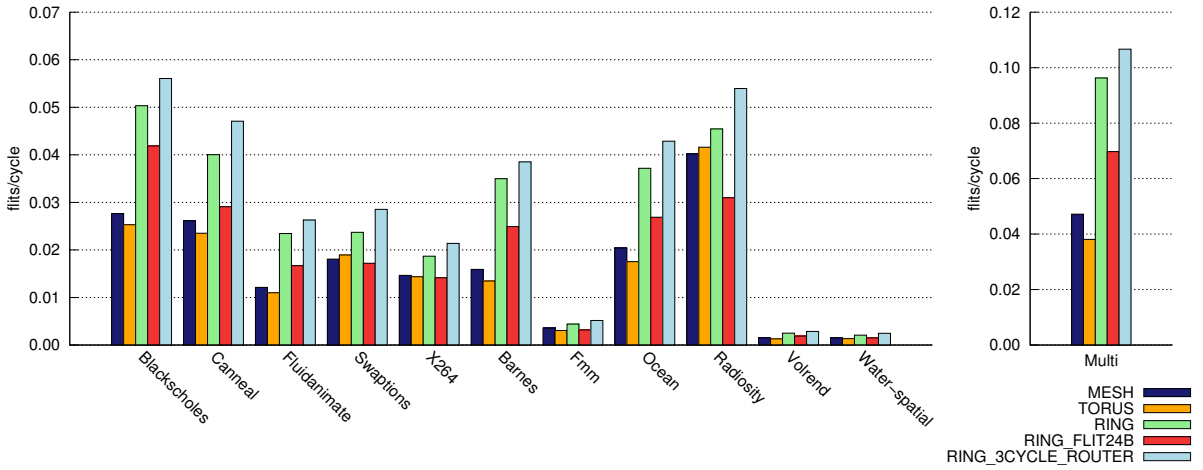
We have already seen that the interconnection network is not heavily used, but it is also important to notice that the use is not uniform. Traffic is unevenly distributed in the interconnect, which means that some resources will be needed more often than others. In this section, we are only presenting results for `blackscholes` among all the parallel applications. We have results for all the other applications but we will omit them due to space constraints. The same conclusions are extracted from all parallel applications. We will also be focusing on results for a 64-core chip, since results are more interesting with a higher core count. Conclusions still hold for 16-core configurations.

Figures 5.3 and 5.4 depict a heat map of injected flits per cycle for each node and link utilization for `blackscholes` executed on 64 cores. The distribution of injection flits is the same regardless of the topology. Values are usually smaller for the rings because a very similar amount of flits gets injected in a much longer period of time. Besides, for the ring with higher bandwidth (24B flits), flits are bigger so we need less flits to send the same amount of information. We can clearly see that some nodes inject more flits than others. Parallel workloads usually have a master thread that drives the execution and distributes work to other threads, which might not be used uniformly. In this case, judging from the heatmaps, we could say that the master thread is located in the bottom left corner of the chip. We can also see that link utilization is higher around the nodes with higher injection rates. Also, link utilization is higher in the ring topologies, since there are less links to transport the same amount of information. The torus wastes more resources since it is the topology where the highest number of flits get injected per cycle, but still has the lowest link usage. Even though we see different patterns in other applications, the conclusions we draw from them are exactly the same.

Figures 5.5 and 5.6 show the same plots for the execution of a multiprogrammed workload. In this case, we see four clear hotspots in the injection pattern in the edges of the chip. Those are the tiles where the memory controllers are located. Apart from that, the rest of ideas we introduced

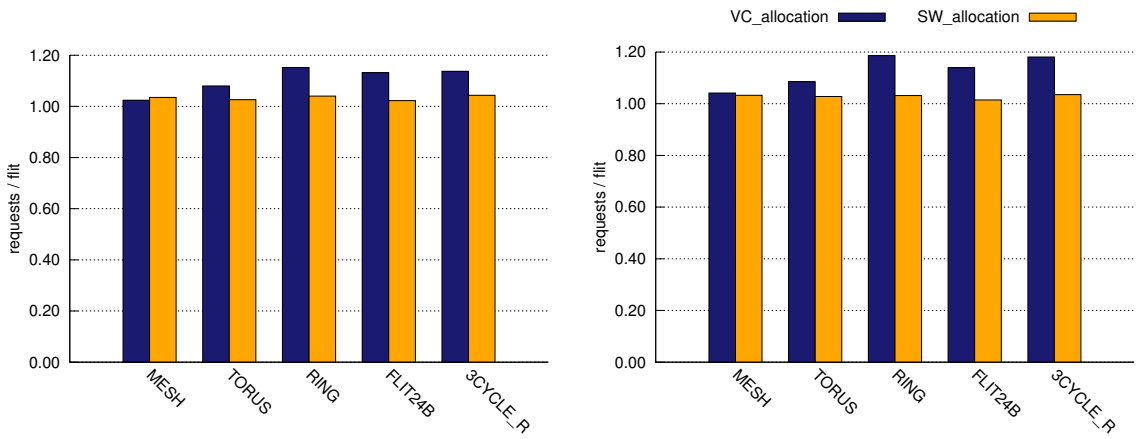


(a) 16 cores



(b) 64 cores

Figure D.3: Average link utilization in flits/cycle for 16 and 64 cores. We distinguish parallel applications (left) and multipgrogrammed workloads (right).

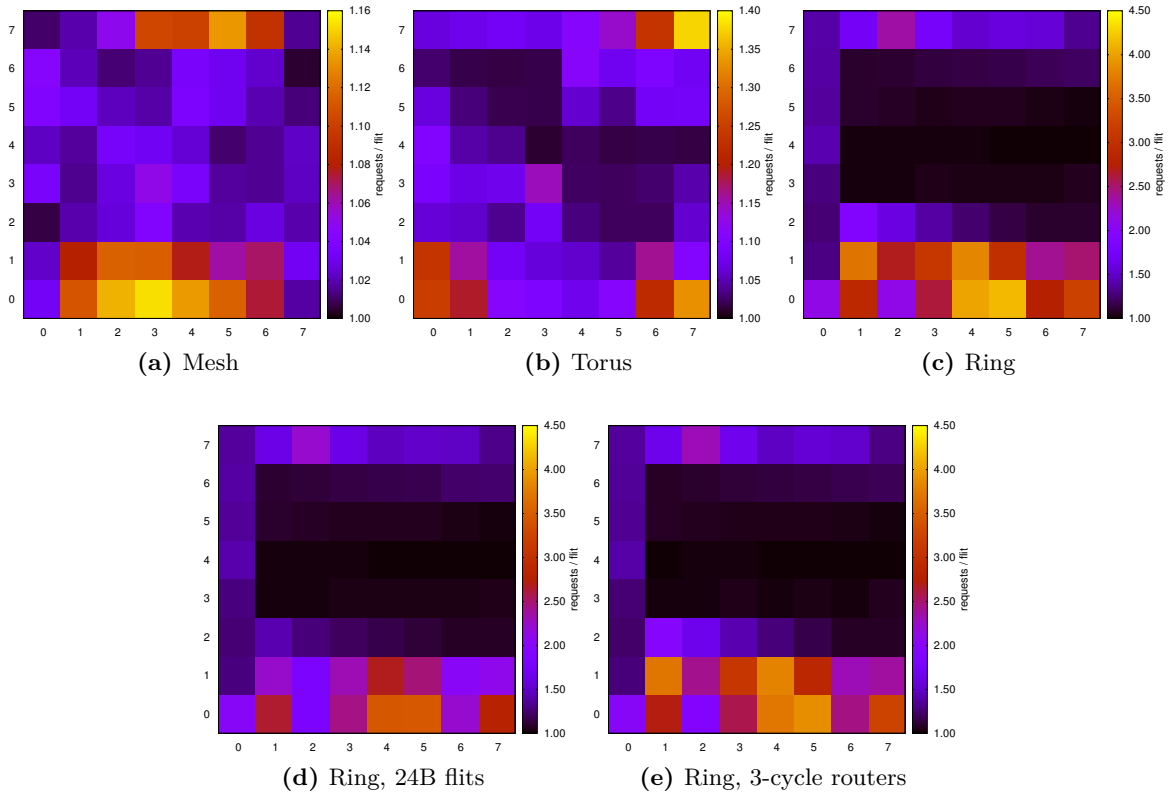


(a) 16 cores **(b)** 64 cores

Figure D.4: Requests per flit for VC allocation and switch allocation for 14 and 64 cores. The ideal value is 1 request per flit, which means messages did not get stalled.

for parallel workloads are still valid. In the mesh topology we can also see that links are more used in the center of the chip, which is the characteristic behaviour for this topology with uniform traffic. Probably, the memory controller location increases link usage in the center of the chip.

We also noticed that, although average congestion is very low, there are some routers that are highly congested and some others that are not congested at all. In Figures D.5 and D.6 we can see the average number of times each flit had to request virtual channel allocation at each router for `blackscholes` and the multiprogrammed workload, respectively. We clearly see that arbitration fails much more often in certain parts of the chip, coinciding with the injection and link utilization hotspots. The differences are especially big for `blackscholes`, where flits need to repeat requests an average of 3 or 4 times in the ring topologies, but they still appear in all the other executions. Again, we see that differences are much larger for the rings, since there are less resources.

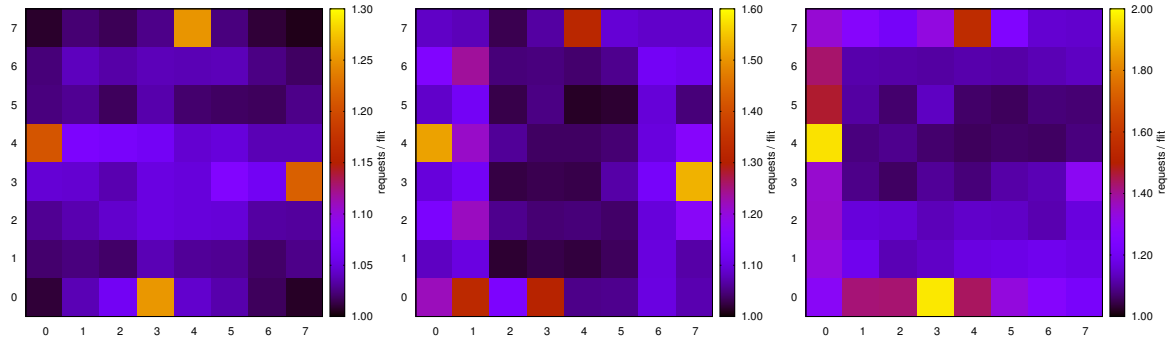


(a) Mesh **(b)** Torus **(c)** Ring

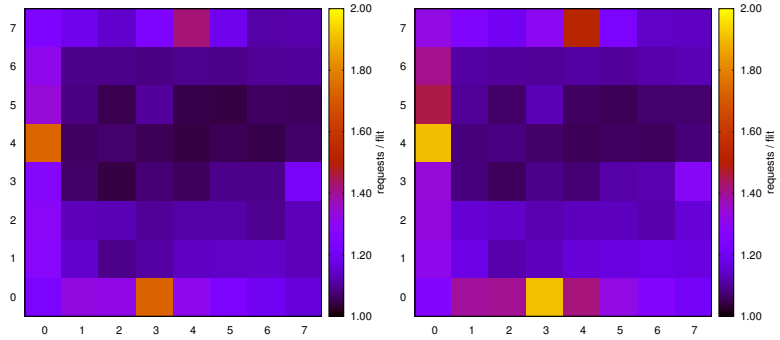
(d) Ring, 24B flits **(e)** Ring, 3-cycle routers

Figure D.5: Virtual channel allocation requests per flit for the `blackscholes` application simulated on 64 cores. Note that scales are different.

These results point out that the network is not being uniformly used. We could have an heterogeneous network where some tiles had more resources and move the threads that need them to those locations. That way, we would be saving power in the *lighter* parts of the chip and would not waste so many resources. Mishra *et al.* already introduce this idea, but they only apply it statically knowing that the center of the chip is usually used more often in mesh topologies [34]. As we have seen, this is not always the case.



(a) Mesh **(b)** Torus **(c)** Ring



(d) Ring, 24B flits **(e)** Ring, 3-cycle routers

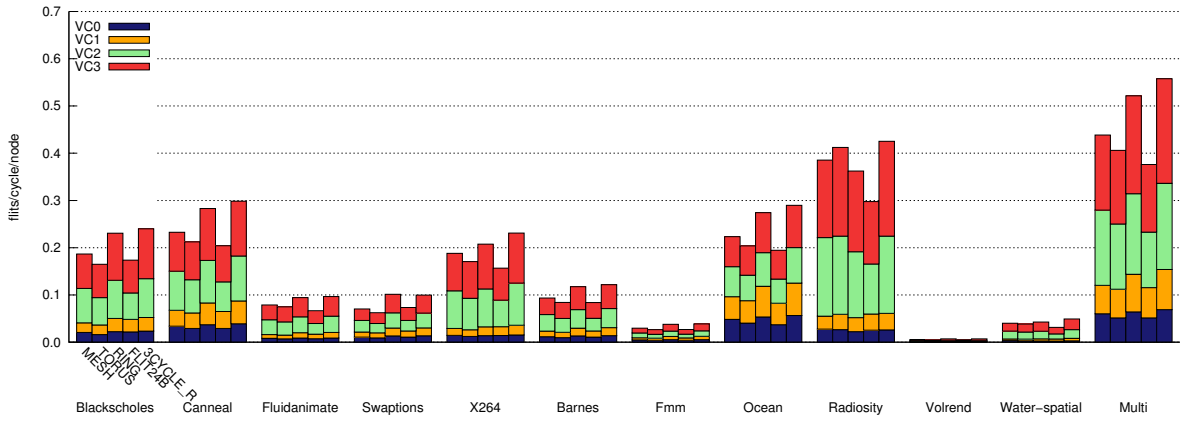
Figure D.6: Virtual channel allocation requests per flit for the multiprogrammed workload simulated on 64 cores. Note that scales are different.

D.3. Type of Traffic Traversing the Network

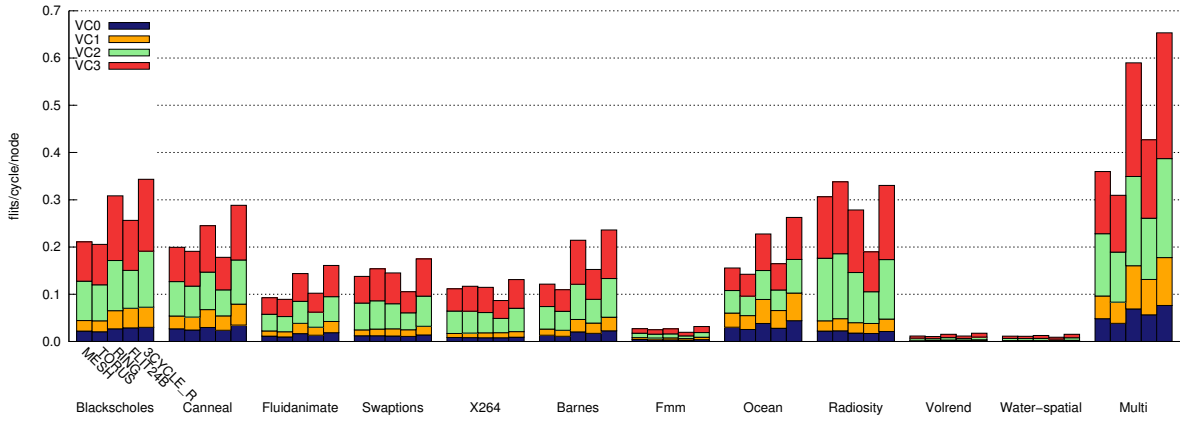
There are different types of messages traversing the network. We need to send control messages, which are one flit long, and data messages, which are five flits long (or three in our ring with higher bandwidth). Traffic can also be classified in requests and replies. Usually, requests are control messages (except for writebacks) and replies are data messages (except for acknowledgements). Figure D.7 shows the use of each virtual network in flits per cycle per node. As we explained in Chapter 3, virtual channels 0 and 1 compose a virtual network used for requests and virtual channels 2 and 3 compose another virtual network used for replies. We see that VN0 is used much less than VN1. That is because messages travelling on VN0 are mostly made up of a single flit, while the ones in VN1 usually have 5 flits. This agrees with Seiculescu *et al.* and Lodde *et al.*, that based on this idea proposed architectures with two separate dedicated networks tuned for the necessities of the traffic they would be used for [39, 28]. We can also see that there is a balanced use of the virtual channels that compose each virtual network, regardless of the topology. This shows that the load balancing we implemented in our deadlock avoidance method introduced in Chapter 3 is working properly.

The virtual channels in the ring are usually used more frequently than in the other topologies because the same number fo flits have to flow through less links and virtual channels, increasing the router congestion. For the ring with 24B flits, the number of flits is smaller because they are bigger and, therefore, we need less filts per message. The cases where the virtual channels in the torus are used more frequently are due to the same amount of traffic traversing the network in a

smaller number of cycles. The multiprogrammed workloads consistently use the virtual channel buffers more than the parallel applications. This is in line with their higher injection rate.



(a) 16 cores



(b) 64 cores

Figure D.7: Average virtual channel load for 16 and 64 cores. Values are represented in flits per cycle per node, in order to compare between the 16 and 64-core architectures. Bars are broken into the load for each of the four virtual channels. The rightmost bars correspond to the multiprogrammed workloads; the rest are parallel applications.

There are also different types of traffic depending on the elements that are communicating: two L1 caches, an L1 and an L2 or an L2 with a memory controller. The number of messages that traverse the network is an effect of the cache miss rates. Figure D.8 shows the number of messages that traverse the network distinguishing the type of communication. Traffic between L1 caches happens rarely, only when an L1 needs to send its private data to another L1. The most common type of traffic happens between the L1 and L2 caches, which happens with every L1 cache miss and writebacks. Communication between the L2 and main memory happens due to L2 cache misses and writebacks of dirty lines from L2. For the multiprogrammed workloads, there is more traffic between the L2 cache and main memory than in parallel workloads, which is consistent with the higher congestion on the tiles with memory controllers we saw in the previous section. In the multiprogrammed workloads, especially for 64 cores, we can see big differences in

the number of messages exchanged. This is because we are executing a fixed number of cycles. Since the ring topologies are slower, less amount of work gets done in the same time and less messages are sent into the network.

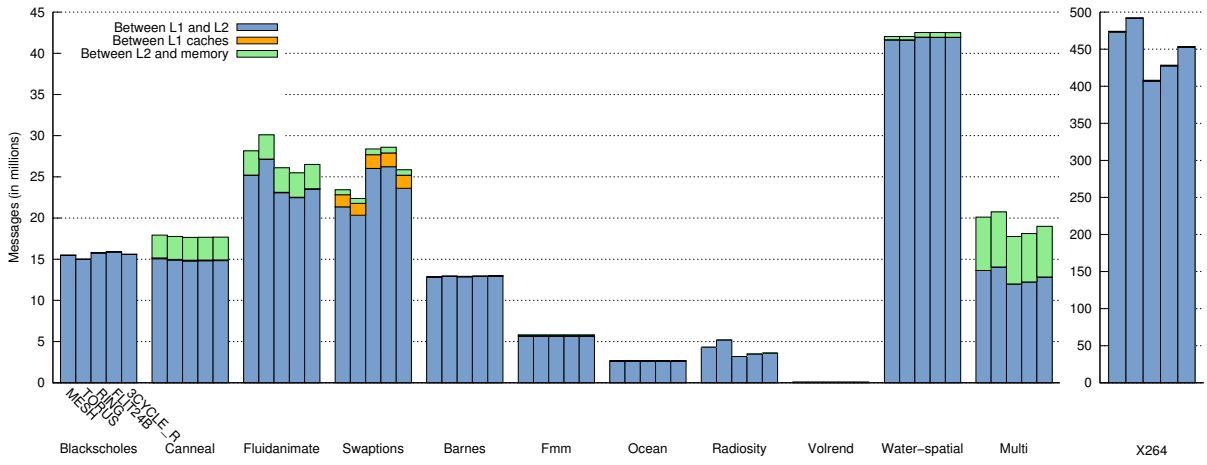
It is interesting to study the differences between the 16-core the 64-core architectures for the parallel applications. In this case, the same amount of work is executed for every application, though it takes longer with some network configurations. The L2 cache is four times bigger with 64 cores, which means there will be less replacements and, therefore, less communication with main memory. This can be checked with `canneal` and `fluidanimate`, where the number of messages exchanged between the L2 and memory gets divided by four when going from 16 to 64 cores. The amount of traffic between the L1 and L2 is slightly bigger in 64 processors. We also have more L1 cache total space, but we also have more cores that will need to communicate with each other.

With most applications, there are no significant differences between topologies in the number of messages exchanged. However, in `fluidanimate`, `x264` and `ocean` there are significant variations in the amount of traffic between L1 and l2, especially in the 64-core configurations. Faster topologies, like mesh and torus, have the side effect of increasing the messages between first and second level caches. We have noted that the difference comes from a much higher number replacements in the L1, which is probabily due to the use of synchronization. Different communication latencies may originate modifications in the ordering of the coherence messages in synchronization phases of the applications. We will analyse this in more detail as part of our future work.

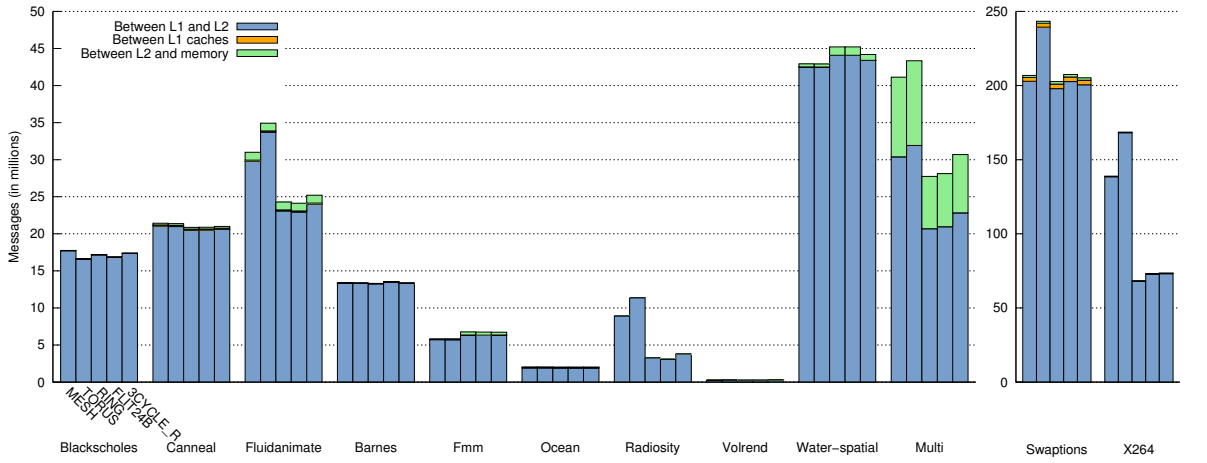
D.4. Area and Power

Figure D.9 represents the area needed by the interconnection network and the L1 and L2 caches. For the interconnect, we show the area for each one of the topologies, which was obtained with Orion. It is broken down into router and link area. The link area is an upper bound for the real value, since we have accounted for wires and repeaters and part of the wires may be router over logic. The torus is the topology with larger area requirements and the ring the one with the lowest. The ring with higher bandwidth was designed so that routers would have the same area as the torus routers. The area for the ring with 3-stage routers is the same as the area for the original ring. If we compare the area used by the interconnect and the area used by all the caches, also depicted in Figure D.9, we see that they are in the same range, which shows that the impact of the interconnect on the chip area is as important as that of the L1 caches. This can be seen more clearly in the piecharts of Figure D.10. For the interconnect, we have used the area of the mesh toplogy, since it was between the values for the ring and torus.

Figure D.11 depicts the energy expended by the interconnect for 16 and 64 cores. These values are calculated with Orion based on the use of the interconnect recorded during simulations. We notice that the ring with increased bandwidth expends much more energy than the original one, and even more than the torus. The consumption of the mesh and torus is quite similar. For 16 cores, the original ring and the one with 3-cycle routers need less energy because routers are simpler and there are less links. In the 64-core architecture, there are some parallel applications for which all the ring topologies expend more energy. This is due to the much larger execution time and higher contention in the interconnect. If parallel applications had perfect scaling, static energy would stay constant when going from 16 to 64 processors because we would multiply by four the number of resources but divide execution time by four. Dynamic energy should



(a) 16 cores



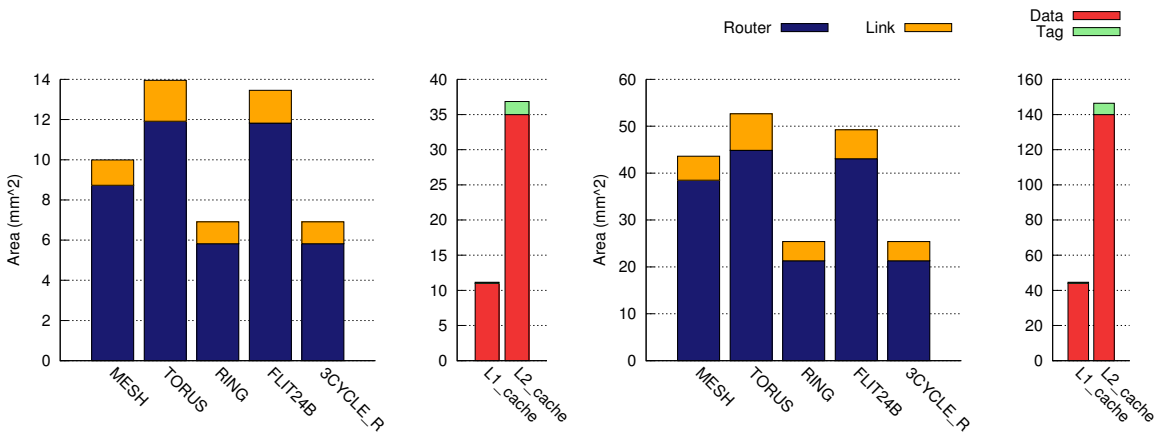
(b) 64 cores

Figure D.8: Traffic between each type of element of the memory subsystem for 16 and 64 cores, meassured in total number of messages (note that some messages are one flit long and others are divided in five or three flits). Note that applications have been rearranged to adapt the scales for better visualization. The rightmost bars of the big plots correspond to the multiprogrammed workloads; the rest are parallel applications.

stay constant. As we can see, energy increases with the number of processors, which means the applications do not scale very well. For the multiprogrammed workloads, where simulation time is constant, we see that both static and dynamic energy are multiplied by four when moving up to 64-core architectures.

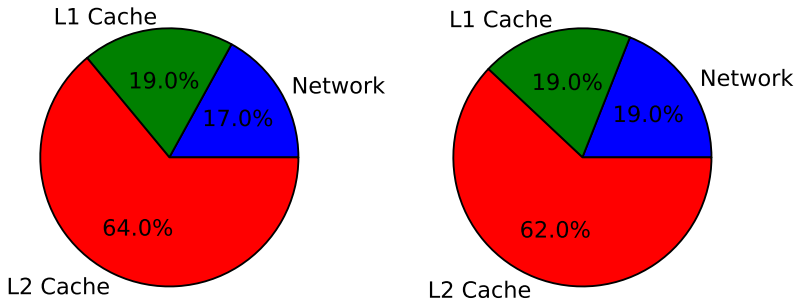
Figure D.12 depicts the energy expended by the L1 caches, which have parallel access (this was described in Appendix B). Figure D.13 shows the energy for the L2 caches, which have sequential access. We notice that applications with higher network energy consumption are the same as applications with higher cache energy consumption, because the use of both resources is correlated. The L2 expends less dynamic energy than the L1 caches, since it is used less often, but more static energy, because it is larger.

Figure D.14 includes pie charts with the energy consumption share of the network and the caches, for 16 and 64 processors. For the network energy, the mesh topology has been used. Also,



(a) 16 cores **(b)** 64 cores

Figure D.9: Network and cache area in mm^2 for 16 and 64 cores. They are aggregated values for all the components of the network and all the caches in the chip. Link area includes both wires and repeaters, so it is an upper bound for the real value, since part of the wires may be router over logic.

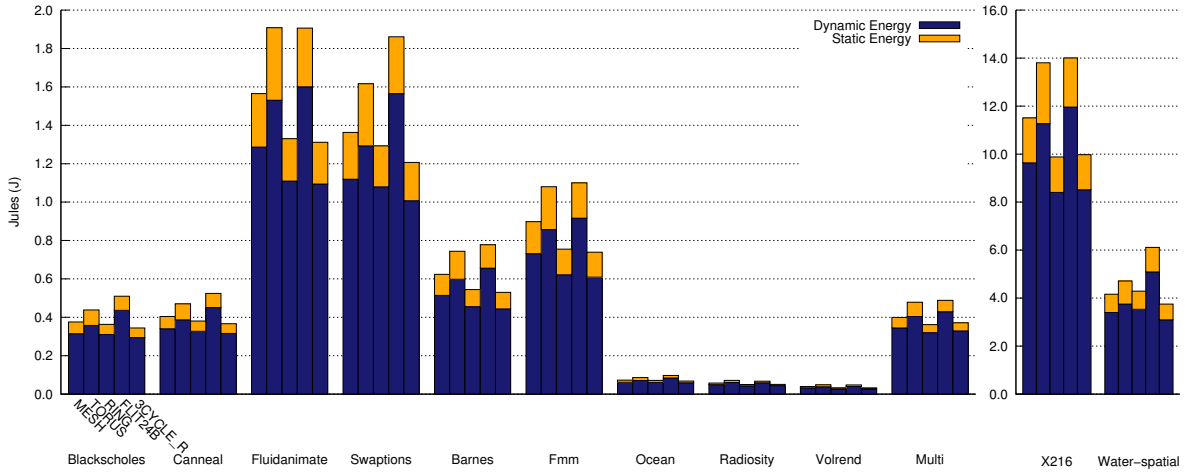


(a) 16 cores **(b)** 64 cores

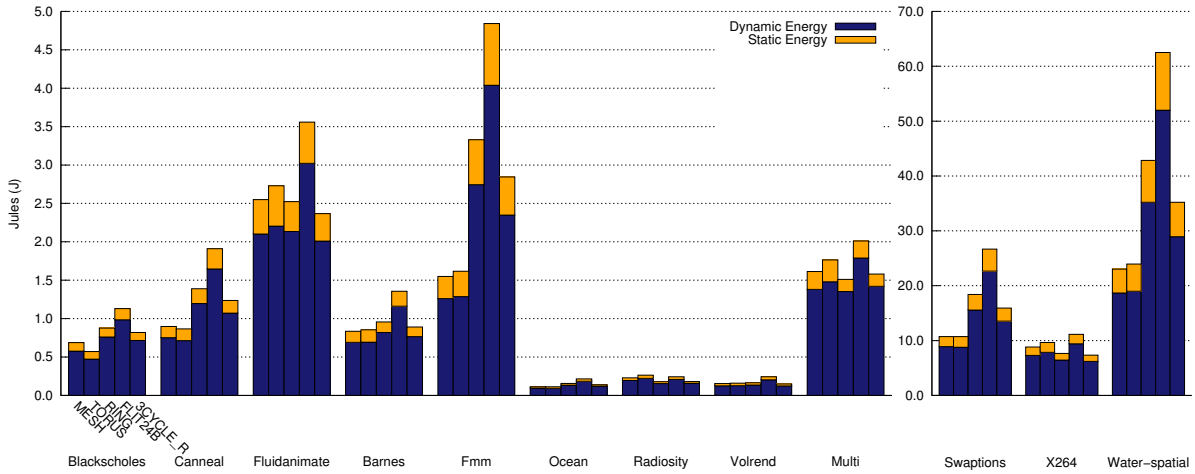
Figure D.10: Pie charts comparing network and cache area for 16 and 64 cores. For the network area, the mesh topology has been used.

the energy expended in clock propagation in the network has been removed to make a more accurate comparison, because the results for the cache did not include it. In parallel workloads, the energy used by the interconnect is approximately the same as the energy used by the L1 and L2 caches combined. For the multiprogrammed workloads, the share of the network is higher, which can be explained by the larger amount of accesses to main memory we already discussed in Sections D.2 and D.3. This results ratify the important share the interconnection network has in the total power consumption of the system.

Figure D.15 shows the energy expended at each component of the router and the links. We can see that the distribution of the energy among the components does not change with different topologies or type of workload. The propagation of the clock consumes most of the energy, followed by the links, the input buffers and the crossbar. The consumption of the allocators is negligible.



(a) 16 cores

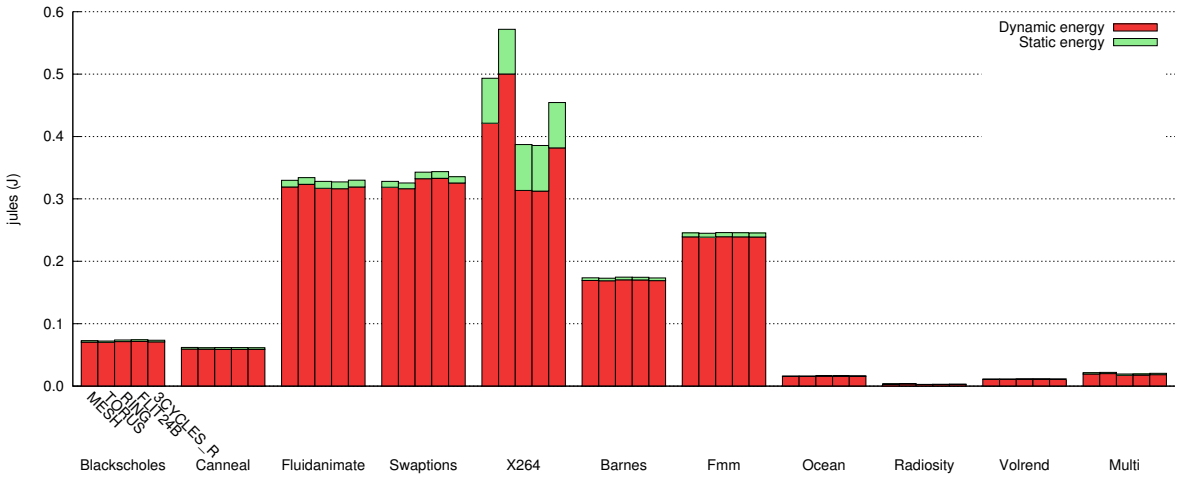


(b) 64 cores

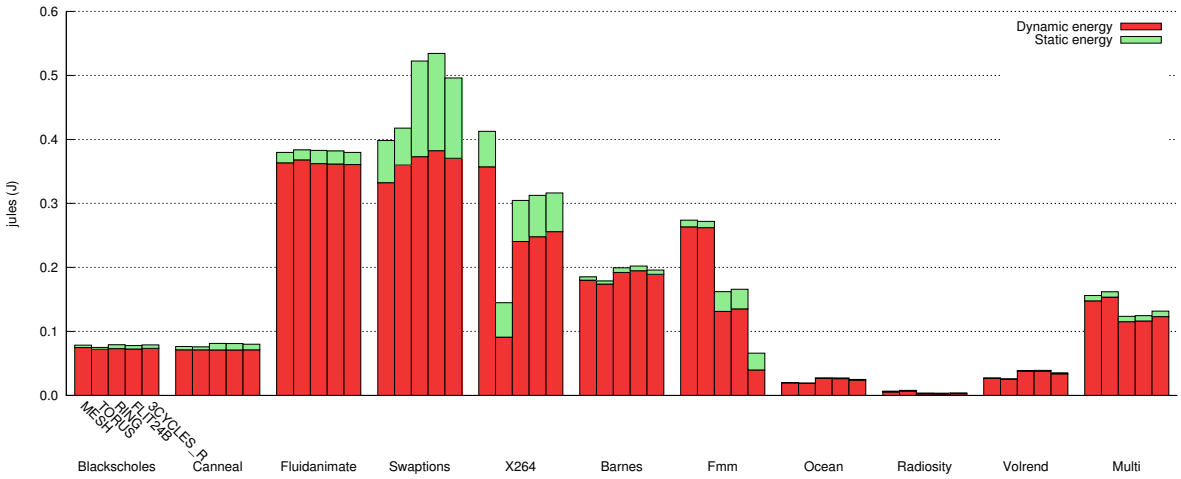
Figure D.11: Energy expended by the interconnection network for 16 and 64 cores, expressed in jules. Bars are broken into dynamic and static energy. Note that applications have been rearranged to adapt the scales for better visualization.

D.5. Topology Selection

When making design choices for future architectures we need to consider performance, power and area. When executing parallel applications on a multicore architecture, we are interested in reducing latency. Therefore, we are calculating energy-delay, which is commonly used to account for both energy consumption and performance improvements. This allows us to visualize the tradeoff and decide if the performance increase we are getting is worth the higher consumption. Using chip multiprocessors to run multiprogrammed workloads, we want to increase our throughput. For that reason, we are going to use energy per instruction (EPI), which indicates the quality of the system in terms of throughput by modeling how much energy is needed for the execution of each instruction. For these workloads, we ran the simulations for a fixed number of cycles (as we explained in Chapter 4, so the energy-delay metric would not be representative. We are considering only the energy expended by the interconnection network since it is the part of the system we are comparing and we already saw that cache energy does not vary among interconnect configurations.



(a) 16 cores

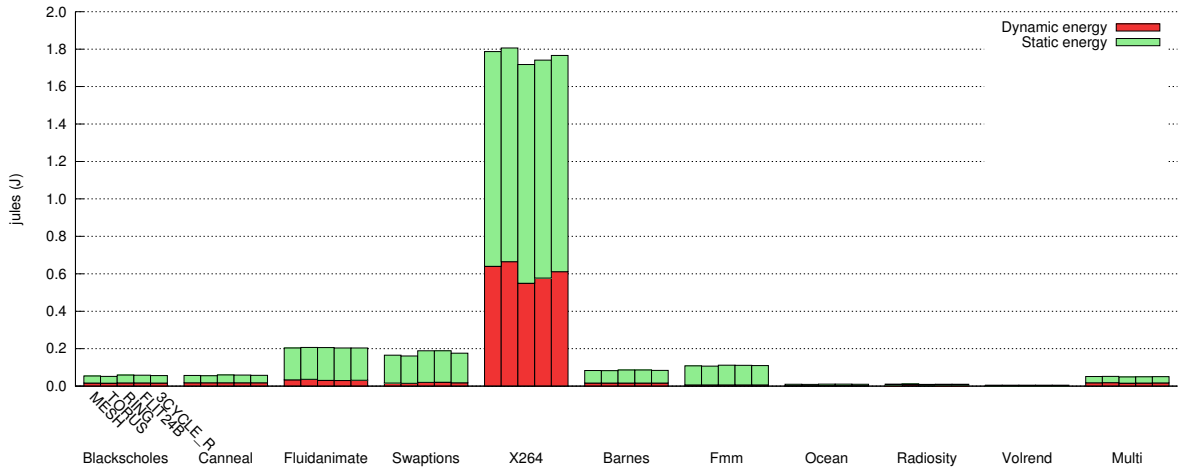


(b) 64 cores

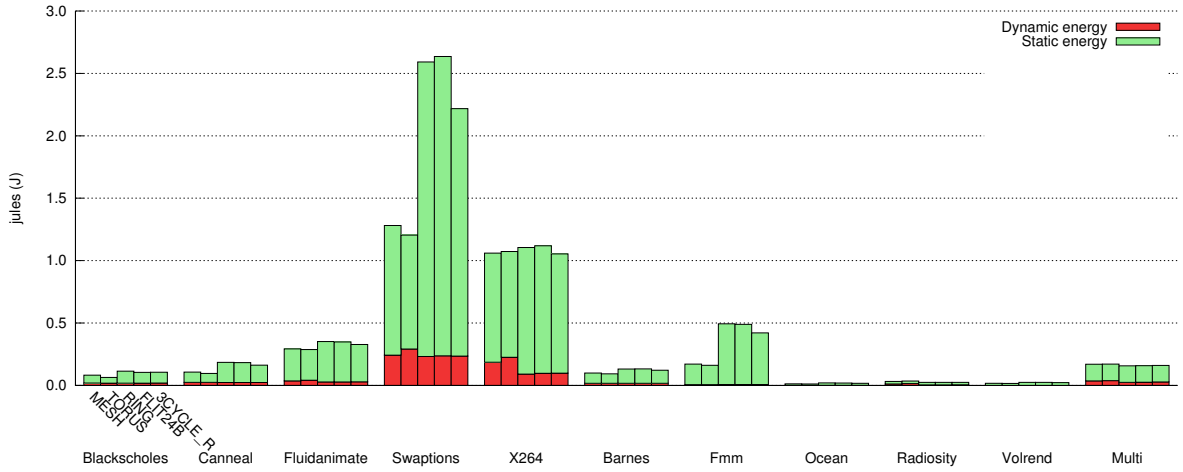
Figure D.12: Energy expended by the L1 cache with 16 and 64 processors, broken into dynamic and static energy. `Water-spatial` has been left out of this graph because the counters used for the calculation of cache energy overflowed due to large execution time. The rightmost bars of the big plots correspond to the multiprogrammed workloads; the rest are parallel applications.

Figure D.16 includes both metrics explained above for 16 and 64-core architectures. When using 16 cores, we are getting the best results with the ring topology with 3-cycle routers, which is similar to the original ring topology for some applications and close to the mesh in most cases. However, if we focus on our 64-core architecture, we most often get the best results with the torus, very closely followed by the mesh. This is because, as we saw in Section D.1, performance drops much more significantly for the ring topologies when we have a higher core count due to the number of hops. In those cases, the increase in energy consumption is justified by the large benefits we get in performance.

In Figure 5.2 we represent area versus energy-delay or EPI, so as to make a decision based on the three main aspects we should consider. For parallel workloads, we are adding up the energy-delay values of all the applications as if they were all executed back to back. We would like to have a configuration with small area and small energy-delay, that is, be in the bottom left corner of the graphs. For the 16-core system, we would clearly choose the ring topology with



(a) 16 cores



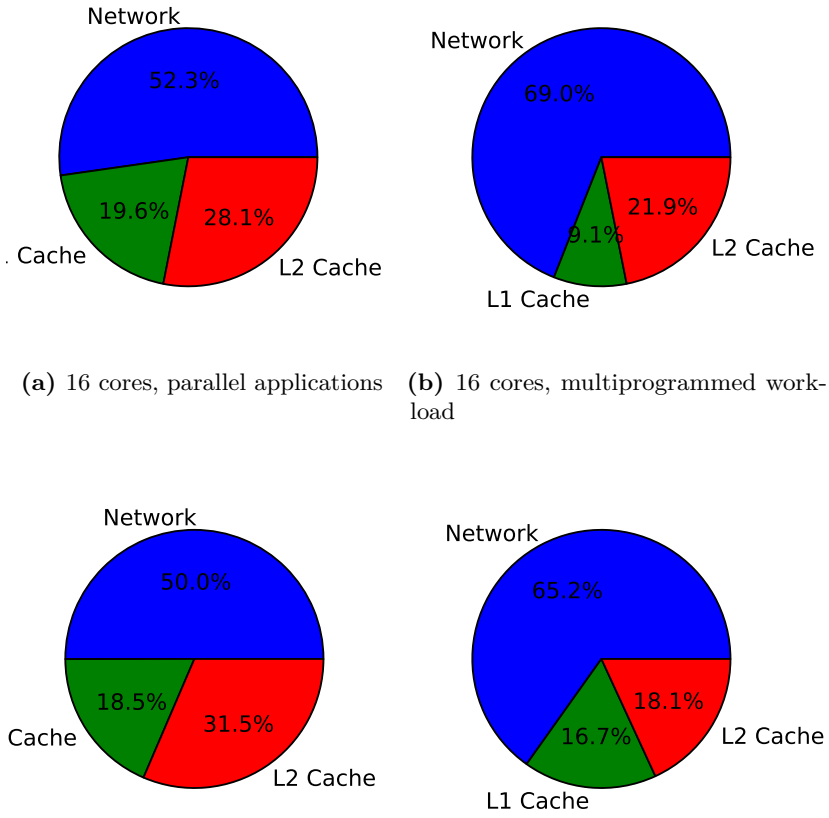
(b) 64 cores

Figure D.13: Energy expended by the L2 cache with 16 and 64 processors, broken into dynamic and static energy. `Water-spatial` has been left out of this graph because the counters used for the calculation of cache energy overflowed due to large execution time. The rightmost bars of the big plots correspond to the multiprogrammed workloads; the rest are parallel applications. Note that scales are different.

3-cycle routers, which gave us the best energy-delay values and has a very small area. With 64 cores, With 64 cores, we could draw a line between the 3-cycle router ring and the mesh topology to represent the Pareto optimal points. Anything above that line would be suboptimal. We see mesh and torus very close with the lowest energy-delay values, but the torus needs a much larger area. Ring topologies still have smaller area, but performance has dropped considerably. In this case, we would choose the mesh topology to get the best tradeoffs. Conclusions are applicable to both parallel and multiprogrammed workloads.

D.6. Conclusions

Firstly, we have seen that a ring topology with 3-cycle routers performs well enough for 16-core chips, while having the least power consumption and area requirements. For 64-core architectures, a ring performs much worse than a mesh and a torus due to the large number of



(a) 16 cores, parallel applications

(b) 16 cores, multiprogrammed workload

(c) 64 cores, parallel applications

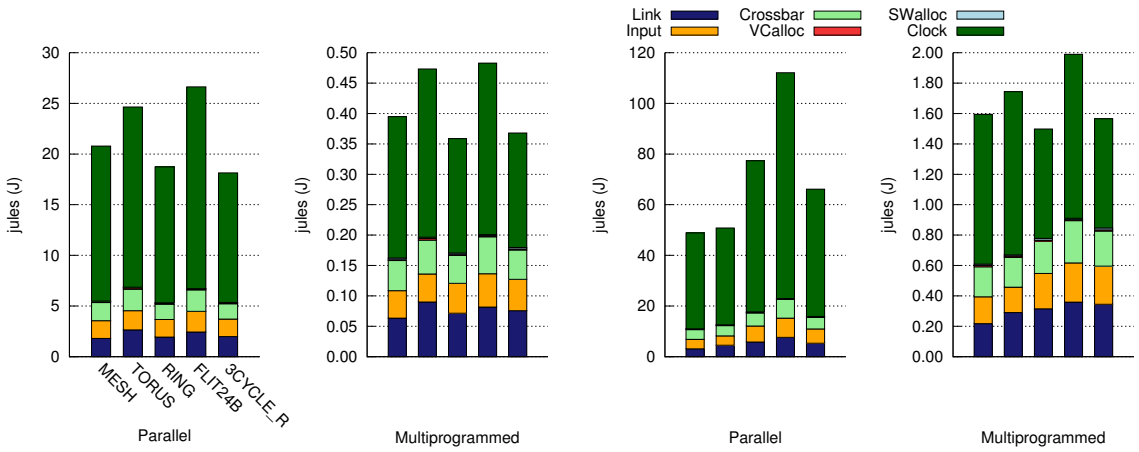
(d) 64 cores, multiprogrammed workload

Figure D.14: Pie charts comparing network and cache energy for 16 and 64 cores, distinguishing between parallel and multiprogrammed workloads. We have added up the energy expended by each application, as if they were all executed back to back. For the network energy, the mesh topology has been used. The energy expended in clock propagation in the network has been removed because the results for the cache did not include it.

hops needed to traverse the network. The torus has the best performance, but the mesh would be the best choice considering area and energy costs. We have verified the results presented in previous work that state that performance depends mostly on the number of hops messages need to traverse the network. We have also ratified that the network is not congested at all, in fact, there are parts of the network that are idle most of the time.

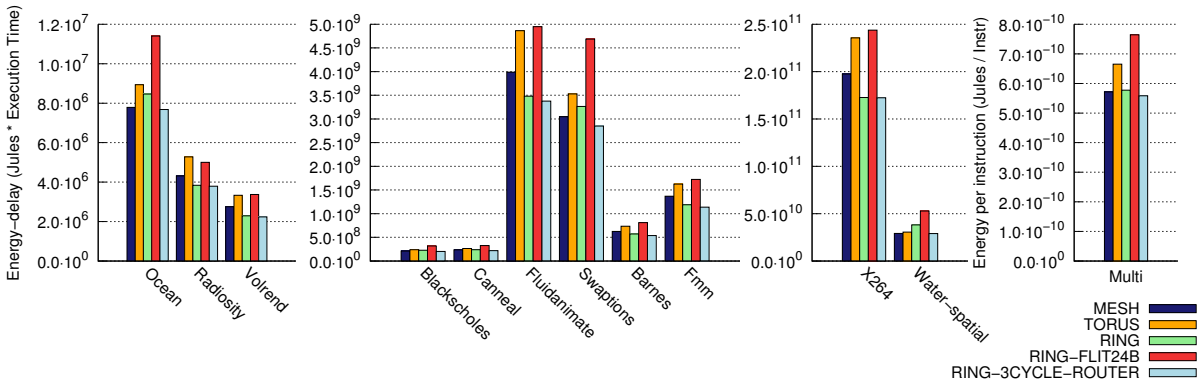
Our most significant contribution is related to the distribution of traffic on the network. We have seen that traffic is not uniformly distributed on the network and that the tiles with higher injection rates vary with the applications. As far as we know, there is no previous research where this behaviour has been noted.

We have also explained that there are different types of traffic traversing the network and how they influence the way resources are used. We have seen that, in some cases, the topology has an effect on the number of messages injected into the network.

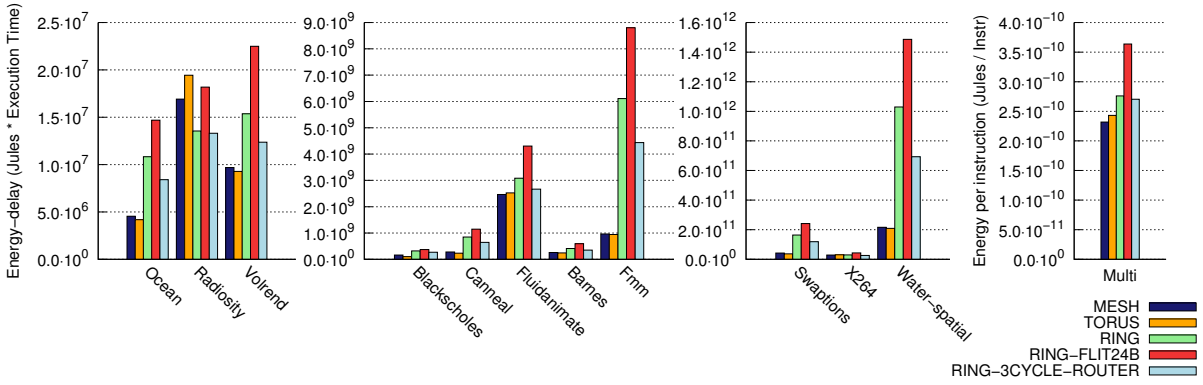


(a) 16 cores **(b)** 64 cores

Figure D.15: Network energy expended by the interconnect for 16 and 64 cores, broken into each one of its components. We sepparate parallel and multiprogrammed workloads. Note that scales are different.



(a) 16 cores



(b) 64 cores

Figure D.16: Tradeoffs between energy and performance. Energy-delay is used for parallel applications and energy per instruction, for multiprogrammed workloads. In both cases, the lower, the better. Note that applications have been rearranged to adapt the scales for better visualization. The rightmost plots correspond to the multiprogrammed workloads; the rest are parallel applications.

Appendix E
Research paper: Characterization of Interconnection Networks in CMPs Using Full-System Simulation

This appendix presents a paper that sumarizes this work and will be submitted to the *Interconnection Workshop on Network Architectures: On-Chip, Multi-Chip*. The workshop will be co-located with the HIPEAC conference that will take place in 2013 in Berlin.

Characterization of Interconnection Networks in CMPs Using Full-System Simulation

ABSTRACT

In modern computer architecture systems, chips are composed of several processors and a significant amount of memory. The importance of the interconnect is growing as the number of nodes integrated in a chip increases, because communication among nodes can become a bottleneck and slow down performance improvement. It also contributes with a substantial share to power consumption and chip area. We compare the behaviour of three topologies: bidirectional ring, mesh and torus, and include two aditional ring topologies: one with increased bandwidth and another with reduced-pipeline routers. We carefully model all the components of the system mentioned earlier using full-system simulation. We demonstrate that a ring topology with 3-cycle routers performs well enough for 16-core chips. For 64-core architectures, rings perform much worse than the mesh and torus due to the large number of hops needed to traverse the network. The torus has the best performance, but the mesh would be the best choice considering area and energy costs. Congestion in the network is not an issue and traffic is highly unbalanced, causing some parts of the network to be idle most of the time.

Keywords

Computer architecture, chip multiprocessor, interconnection networks, topologies

1. INTRODUCTION

Nowadays, chips are composed of several processors and a significant amount of memory. A popular trend in the organization of general purpose chips consist on interconnecting several nodes (usually called tiles), each of them with a processor (core) and one or more levels of shared and/or private memory caches. Nodes communicate through an interconnection network that allows any pair of nodes to exchange information. If the current trend continues, the scale of integration, along with the emergent 3D stacking technology, will allow us to exponentially increase the number of nodes in a chip, reaching hundreds or even thousands of cores in less than ten years. So that this technology evolution can be translated into an improvement on performance, the capacity of communication among tiles should scale in the same proportion.

At these time, there are very few studies that model in detail the set of processors, interconnection network and memory hierarchy. Besides, analysis that focus on interconnection networks are usually performed simulating synthetic traffic or application traces that do not capture the behaviour of a real execution. We analyse the behaviour of real applications on the network, carefully modeling all the components mentioned earlier. This allows us to study the effect of the interconnection network configuration on the whole system and the interactions between the memory subsystem and the interconnect. Specifically, we compare the performance of three topologies (bidirectional ring, mesh and torus), simulating a chip multi-processor (CMP) with 16 and 64 cores. Our aim is to extract meaningful conclusions that will guide future research on this field.

The remainder of this document is organized as follows: section 2 presents the state of the art in interconnection network research; section 3 describes the architecture of our system; section 4 explains the methodology; 5 summarizes our results and section 6 concludes the report.

2. RELATED WORK

There are many papers that propose alternatives to the most commonly used router architectures, topologies and flow control methods, but none of them model the impact of their contributions by running real programs on a full system. Among that research, we highlight the following: Mishra *et. al.* propose an heterogeneous on-chip interconnect that allocates more resources for routers suffering higher traffic but they only get good results with a mesh topology [19]; Koibuchi *et. al.* detect that adding random links to a ring topology results in big performance gains, although they only experiment with a network simulator [14].

It is worth mentioning another approach to network on chip research. Instead of dealing with classical network issues, there is previous work that tries to improve performance based on the known behaviour of the memory subsystem and the coherence protocol. A selection of recent papers follows. Seiculescu *et. al.* propose to use two dedicated networks, one for requests and one for replies [23]. Agarwal *et. al.* propose embedding small in-network coherence filters inside on-chip routers to dynamically track sharing patterns and eliminate broadcast messages [2]. Krishna *et. al.* propose a system to improve the frequent 1-to-many and many-to-1 communication by forking and aggregating packets to avoid the increment in the amount of traffic when scaling the number of nodes [15]. These studies try to improve the performance of the most commonly used networks, but do not venture with less conventional topologies.

Table 1: Main characteristics of the CMP system.

Cores	16 and 64 cores, Ultrasparc III Plus, in order, 1 instr/cycle, single threaded, 2GHz frequency
Coherence protocol	Directory-based, MESI, directory distributed among L2 cache banks
Consistency model	Sequential
L1 cache	32KB data and instruction caches, 4-way set associative, 2-cycle hit access time, 64B line size Pseudo-LRU replacement policy
L2 cache	Distributed, 1 bank/tile, 1MB per bank, 16-way set associative, 7-cycle hit access time, 64B line size Pseudo-LRU replacement policy shared, inclusive, interleaved by line address
Memory	4 memory controllers, distributed in the edges of the chip (both for 16 and 64-core architectures) 160-cycle latency

There are very few papers which focus on the comparison of interconnection network configurations. Balfour and Dally present an analysis of how different topologies affect performance, area and energy efficiency [3]. However, they do not model the memory subsystem, they only use synthetic traffic patterns and they fail to include a simpler topology like the ring. Sanchez *et. al.* explore the implications of interconnection network design for CMPs, but they focus on more complex topologies, such as the fat three and the flattened butterfly [22]. Their main results point out that the main parameter that influences performance is the number of hops the messages will need to get from one tile to another in the network. They also highlight the need of a careful codesign of the interconnection network and the cache hierarchy. This necessity has also been noted by Kumar *et. al.*, but their research does not go above 16 cores [16].

In this work we present an analysis of three topologies with varying degrees of complexity, performance and power and area costs. We perform full-system simulation of real workloads and carefully model both the memory subsystem and the interconnect. Our aim is to extract meaningful conclusions that will indicate the weaknesses of current configurations and guide our future research.

3. CMP ARCHITECTURE FRAMEWORK

This section presents the architecture of the chip multiprocessor we are modeling, including a more detailed description of the interconnection network.

3.1 General description of the architecture

This study focuses on homogeneous chip multiprocessors (CMPs). The system is composed of several *tiles* connected by an interconnection network. On each tile we have a core with a private first level cache (L1) split into data and instructions, a bank of the shared second level cache (L2), a router and we may also have a memory controller. Table 1 summarizes the key parameters of our system. To model the architecture we based our design in other systems with similar characteristics, both from academia research [28, 23, 4] papers and commercial processors such as IBM Power4 [25], Tilera's TILE*Pro*64 [26], Intel 48-core processor [9, 10] and Sun Microsystems' Niagara2 [21].

We are using a directory-based MESI coherence protocol. All the traffic that traverses the interconnection network is a direct consequence of the memory activity, either to move cache lines (instructions or data) to the tile that needs them or for coherence management. That is why it is important to model the caches realistically, even though our main interest is the interconnect [16, 22].

3.2 Interconnection network

We are going to compare three different topologies: mesh, torus and ring. The *2D mesh* is a widespread choice for large-scale CMPs due to its regularity. A *torus* is a mesh in which we add wraparound links so as to reduce the average number of hops between tiles. This topology is the one that will need more resources in area and power. In contrast, we have included a *bidirectional ring*. So as to keep the same organization of the chip following a matrix layout, the ring is built as a hamiltonian cycle. Table 2 summarizes the main characteristics of the three topologies.

The network has packet-switched wormhole credit-based flow control, dimension order routing. We use a pipelined router with four stages: input buffering and routing, virtual channel allocation, switch allocation and switch traversal. So every hop takes a total of five network cycles, including link traversal. Further information about the characteristics of the network can be found in Table 3.

In the ring topology, the number of inputs/outputs to the outside of the tile is reduced to two (as opposed to the four used in mesh and tours), which results in a smaller number of buffers and simpler allocators and crossbar. For that reason, the routers in the ring topology will need a much smaller area. To make use of this idle space, we have also tested a configuration in which we increase the link bandwidth keeping the router area in the ring slightly under the one for the mesh and torus topologies. This has allowed us to have flits of 24 bytes, which will reduce the number of flits needed per message and, therefore, the network latency. Following the same idea, since the complexity of the crossbar is considerably reduced, we have also included tests merging the switch allocation and switch traversal stages, resulting in a 3-cycle router.

4. METHODOLOGY

In this section we introduce the metrics, workloads and simulation environment used in the project.

4.1 Metrics

Our study focuses on the comparison of interconnection network topologies, therefore, we are using several tradditional interconnect-centric metrics. We study node throughput, link utilization, hop count, latency, virtual channel load and

Table 2: Qualitative comparison of the three topologies for a CMP system with N tiles (we assume that N will always be a perfect square). The number of inputs\outputs does not consider tiles with a memory controller, where routers would have one more input and output, or the tiles in the edges of the mesh, where some ports would be left unused. W is the link bandwidth and L is the wire length.

Topology	**inputs/ outputs**	**Bisection BW**	**Max. hops**	**Avg. hops**	**Link length**
2D mesh	6/6	$2W\sqrt{N}$	$2\sqrt{N}$	$2/3\sqrt{N}+1$	L
Torus	6/6	$8W\sqrt{N}$	$\sqrt{N}+2$	$1/2\sqrt{N}+2$	$L\sqrt{2}$
Ring	4/4	$4W$	$N/2+2$	$N/4$	L

Table 3: Main characteristics of the interconnection network.

General	Two virtual networks (requests and replies)
Routers	4-stage pipeline: routing and input buffering, VC allocation, switch allocation and switch traversal Round-robin 2-phase VC/switch allocators 2 VCs per virtual network 5-flit buffers per virtual channel, enough to store a whole message (3-flits per buffer in ring with higher bandwidth)
Links	16-byte flit size (we also include a ring with higher bandwidth with 24B flit size) 1-cycle latency
Technology	32nm, 2GHz frequency, $V_{dd} = 1V$

the number of arbitration requests. These metrics help us gain insight on how each topology behaves with the workloads.

Power consumption is a key factor in the design of new architectures. We include the energy and area costs of the interconnect and compare them to those of the memory subsystem.

We are modeling all the components of the architecture and we want to examine the impact the different network configurations have on the whole system, so we include metrics that that reflect the overall performance and the behaviour of the memory hierarchy. We analyse the execution time of every application with all the configurations to see which one leads to better performance. We also include miss rate results, which are a determinant factor in the pressure imposed on the interconnect.

4.2 Workloads

The chip multiprocessors we are focusing on may be used to execute parallel applications in order to reduce execution time or for multiprogrammed workloads (execution of independent programs on each core), to increase throughput. We are using a selection of shared-memory parallel applications from PARSEC [5] and SPLASH2 [27] and a multiprogrammed workload made up of SPEC2K6 benchmarks [24]. To be fair, we tried to put together a set of applications with varying characteristics in terms of miss rate, communication, traffic and scaling. We are using `blackscholes`, `canneal`, `fluidanimate`, `swaptions` and `x264` from the PARSEC benchmark suite (with the large input except for x264, for which we used medium due to the large simulation time) and `barnes`, `fmm`, `ocean`, `radiosity`, `volrend` and `water-spatial` from SPLASH2. For SPEC2K6, we chose 16 applications with high footprint and working set size (according to [8] and [6]). To build the workload for the 16-core architectures we have executed each application once, binding each one of them to a different core so that no migration occurs. For the 64-core architectures we have used the applications four times. To execute this workload, we first warm up the caches for 200 million cycles and then execute for 500 million cycles.

4.3 Simulation environment

We are using Simics full-system simulator [17] and GEMS [18]. We use a modified version of GARNET, which was tuned to produce the statistics and support the topologies we needed [1]. We carefully model all the components of the system and perform full-system simulation with simple single-threaded cores and directory-based coherence.

To get the area and energy expended by the network we used the circuit modeling tool Orion 2.0 [12, 13]. For the memory hierarchy we used CACTI 6.5 [20]. The technology files in both tools have been matched so as to be able to compare the results obtain from the two models. Some parameters have been taken from the work of Gracia *et. al.* [7] and accuracy has been further improved by approximating our values to the ones used in INTEL architectures [11].

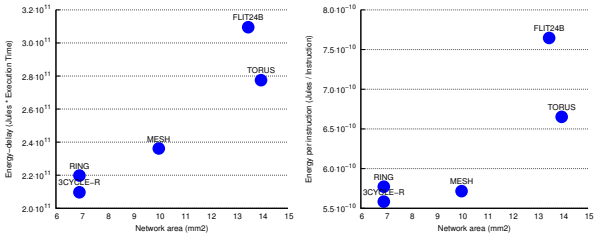
5. RESULTS

This section summarizes the main contributions of our analysis for 16 and 64-core architectures

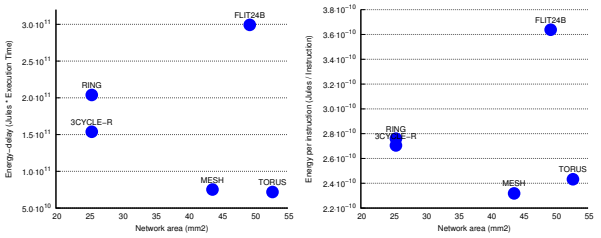
5.1 Network topology comparison

To compare the impact of the network configurations on performance, we are studying the number of processor cycles it takes for the parallel workloads to complete the parallel section. For the multiprogrammed workloads, we check how many instructions get executed in 500 million cycles. When making design choices for future architectures we need to consider performance, power and area. When executing parallel applications on a multicore architecture, we are in-

terested in reducing latency. Therefore, we are calculating energy-delay, which is commonly used to account for both energy consumption and performance improvements. This allows us to visualize the tradeoff and decide if the performance increase we are getting is worth the higher consumption. Using chip multi to run multiprogrammed workloads, we want to increase our throughput. For that reason, we are going to use energy per instruction (EPI), which indicates the quality of the system in terms of throughput by modeling how much energy is needed for the execution of each instruction.



(a) Parallel applications (left) and multiprogrammed workloads (right) executed on 16 cores



(b) Parallel applications (left) and multiprogrammed workloads (right) executed on 64 cores

Figure 1: Area versus energy-delay or energy per instruction for 16 and 64 cores. We also distinguish between parallel applications (left) and multiprogrammed workloads (right). For the parallel applications, they have all been considered together, as if they were executed back to back.

Figure 1 represents area versus energy-delay or EPI for 16 and 64-core architectures, which will help us to make a decision based on the three main aspects we should consider. First, we are going to focus only on the vertical axe, which is representing the energy-delay or EPI. For parallel workloads, we are adding up the energy-delay values of all the applications as if they were all executed back to back. When using 16 cores, we are getting the best results with the ring topology with 3-cycle routers, which is similar to the original ring topology for some applications and close to the mesh in most cases. However, if we focus on our 64-core architecture, we most often get the best results with the torus, very closely followed by the mesh. This is because, as we saw in Section **??**, performance drops much more significantly for the ring topologies when we have a higher core count due to the number of hops. In those cases, the increase in energy consumption is justified by the large benefits we get in performance.

We are now going to consider the area requirements for each topology. The torus is the topology with larger area requirements and the ring the one with the lowest. The ring with higher bandwidth was designed so that routers would have the same area as the torus routers. The area for the ring with 3-cycle routers is the same as the area for the original ring.

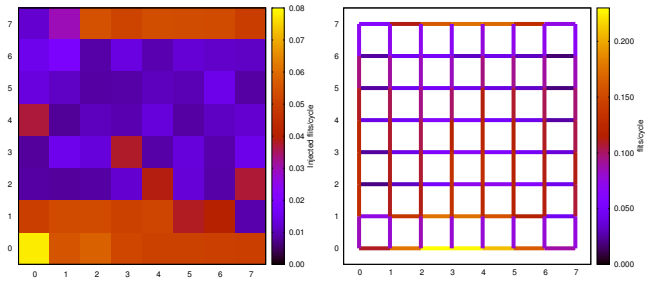
If we look at all the parameters included in the graph, we would like to have a configuration with small area and small energy-delay, that is, be in the bottom left corner of the graphs. For the 16-core system, we would clearly choose the ring topology with 3-cycle routers, which gave us the high energy-delay values and has only very small area. With 64 cores, we see mesh and torus very close with the lowest energy-delay values, but the torus needs a much larger area. Ring topologies still have smaller area, but performance has dropped considerably. In this case, we would choose the mesh topology to get the best tradeoffs. Conclusions are applicable to both parallel and multiprogrammed workloads.

5.2 Non uniform traffic distribution

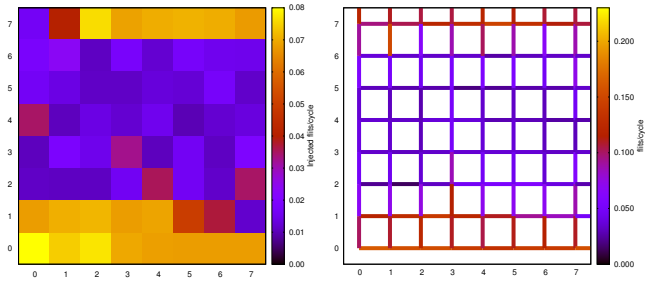
We have analysed the number of injected flits and the link utilization for all our architecture configurations and workloads. We have noted that traffic is unevenly distributed in the interconnect, which means that some resources will be needed more often than others. In this section, we are only presenting results for `blackscholes` among all the parallel applications. We have results for all the other applications, but we will omit them due to space constraints. The same conclusions are extracted from all parallel applications. We will also be focusing on results for a 64-core chip, since results are more interesting with a higher core count. Conclusions still hold for 16-core configurations.

Figure 2 depicts a heat map of injected flits per cycle for each node and link utilization for `blackscholes` executed on 64 cores. The distribution of injection flits is the same regardless of the topology. Values are smaller for the ring topologies because a very similar amount of flits gets injected in a much longer period of time. Besides, for the ring with higher bandwidth (24B flits), flits are bigger so we need less flits to send the same amount of information. We can clearly see that some nodes inject many more flits than others. Parallel workloads usually have a master thread that drives the execution and distributes work to other threads, which might not be used uniformly. In this case, judging from the heatmaps, we could say that the master thread is located in the bottom left corner of the chip. We can also see that link utilization is higher around the nodes with higher injection rates. Also, links are much more used in the rings, since there are less links to transport the same amount of information. The torus wastes more resources since it is the topology where the highest number of flits get injected per cycle, but still has lowest link usage. Even though we see different patterns in other applications, the conclusions we draw from them are exactly the same.

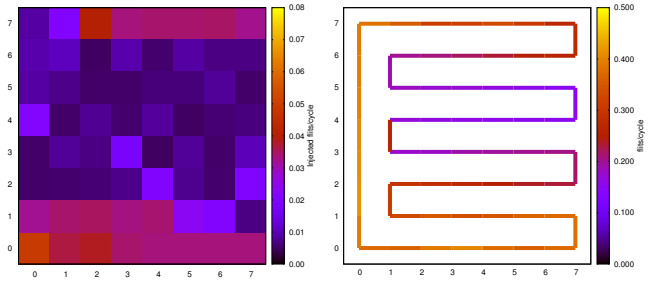
Figure 3 shows the same plots, but for the execution of a multiprogrammed workload. In this case, we see four clear hotspots in the injection pattern in the edges of the chip.



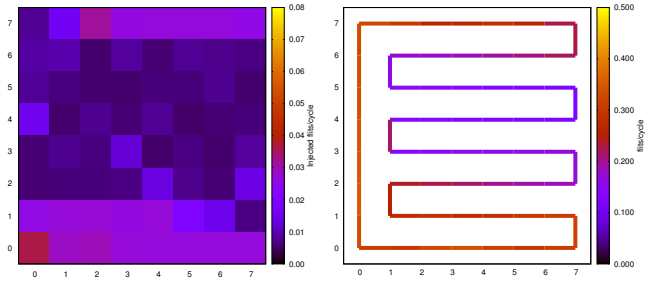
(a) Mesh



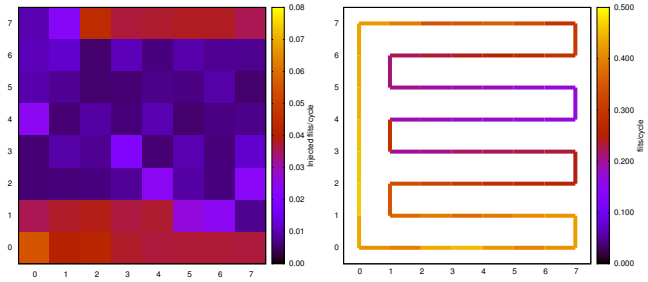
(b) Torus



(c) Ring



(d) Ring, 24B flits



(e) Ring, 3-cycle routers

Figure 2: Injected flits per node (left) and link utilization (right) for the `blackscholes` application executed in 64 cores. For link utilization, each line is the combination of two links, one in each direction. Note that the scale has been kept constant among topologies for injection figures, but not for link utilization. In the torus, links that touch the edges of the chip represent the wraparound links.

Those are the tiles where the memory controllers are located. Apart from that, the rest of ideas we introduced for parallel workloads are still valid. In the mesh topology we can also see that links are more used in the center of the chip, which is the characteristic behaviour for this topology with uniform traffic. Probably, the memory controller location increases link usage in the center of the chip.

These results point out that the network is not being uniformly used. We could have an heterogeneous network where some tiles had more resources and move the threads that need them to those locations. That way, we would be saving power in the *lighter* parts of the chip and would not waste so many resources. Mishra *et al.* already introduce this idea, but they only apply it statically knowing that the center of the chip is usually used more often in mesh topologies [19]. As we have seen, this is not always the case. As far as we know, there is no previous research that introuduces the idea of non uniform traffic derived from the behaviour of applications.

6. CONCLUSIONS

Interconnection networks have a significant influence on system performance, area and power consumption. The actual trend in computer architecture design consists on integrating several cores on a single chip. The network on chip is responsible for the communication between those cores and the caches its design is crucial for guaranteeing improvements in performance.

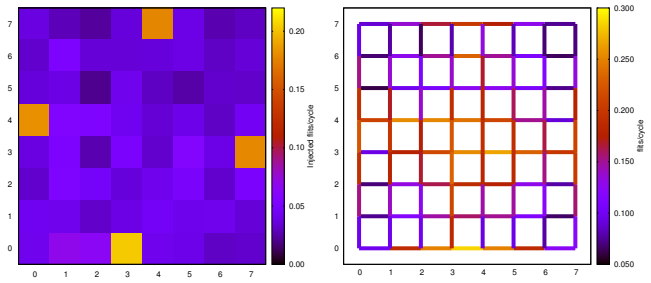
We have compared the behaviour of three network topologies: mesh, torus and ring. We include two additional ring configurations: one with bigger flits and one with 3-cyle routers. We carefully model the processors, memory hierarchy and the network using full-system simulation and executing real applications, including both parallel and multiprogrammed workloads. We include results for CMPs with 16 and 64 cores.

We have demonstrated that performance is highly affected by the choice of the interconnect in 64-core systems. The ring topologies produce much larger execution times due the increased number of hops it takes to traverse the network. The torus has the best performance, but with higher power and area costs. In this case, the mesh would be the best choice. On the other hand, for 16-core chips, differences in performances are not so big and a ring topology with 3-cycle routers offers acceptable performance with the lowest power consumption and area requirements.

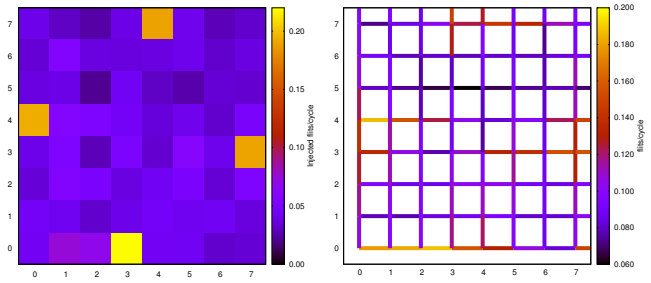
We have seen that traffic is not uniformly distributed on the network and that the tiles with higher injection rates vary with the applications. For multiprogrammed workloads, hotspots are always located in the memory controllers. As far as we know, there is no previous research where this behaviour has been noted.

7. REFERENCES

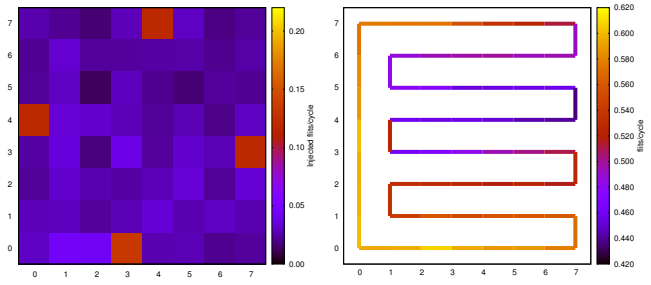
[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha. Garnet: A detailed on-chip network model inside a



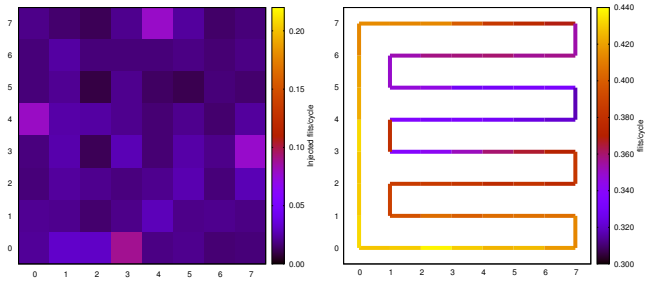
(a) Mesh



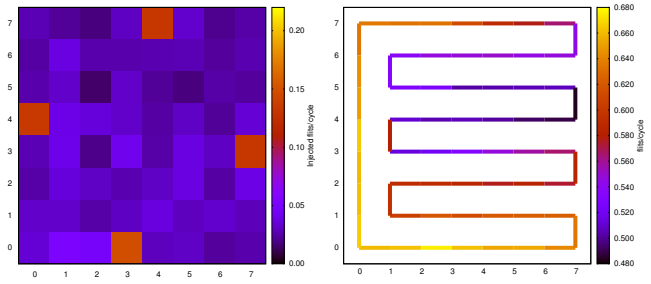
(b) Torus



(c) Ring



(d) Ring, 24B flits



(e) Ring, 3-cycle routers

Figure 3: Injected flits per node (left) and link utilization (right) for the multiprogrammed workload application executed in 64 cores.

full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33 –42, 2009.

[2] N. Agarwal, L.-S. Peh, and N. K. Jha. In-network coherence filtering: snoopy coherence without broadcasts. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 232–243, New York, NY, USA, 2009. ACM.

[3] J. Balfour and W. J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 187–198, New York, NY, USA, 2006. ACM.

[4] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 282–293, New York, NY, USA, 2000. ACM.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[6] D. Gove. Cpu2006 working set size. *SIGARCH Comput. Archit. News*, 35(1):90–96, Mar. 2007.

[7] D. Gracia, G. Dimitrakopoulos, T. Arnal, M. Katevenis, and V. Yufera. Lp-nuca: Networks-in-cache for high-performance low-power embedded processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(8):1510 –1523, aug. 2012.

[8] J. L. Henning. Spec cpu2006 memory footprint. *SIGARCH Comput. Archit. News*, 35(1):84–89, Mar. 2007.

[9] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173 –183, jan. 2011.

[10] Intel. The scc platform overview. 2012.

[11] C.-H. Jan, M. Agostinelli, M. Buehler, Z.-P. Chen, S.-J. Choi, G. Curello, H. Deshpande, S. Gannavaram, W. Hafez, U. Jalan, M. Kang, P. Kolar, K. Komeyli, B. Landau, A. Lake, N. Lazo, S.-H. Lee, T. Leo, J. Lin, N. Lindert, S. Ma, L. McGill, C. Meining, A. Paliwal, J. Park, K. Phoa, I. Post, N. Pradhan, M. Prince, A. Rahman, J. Rizk, L. Rockford, G. Sacks, A. Schmitz, H. Tashiro, C. Tsai, P. Vandervoorn, J. Xu, L. Yang, J.-Y. Yeh, J. Yip, K. Zhang, Y. Zhang, and P. Bai. A 32nm soc platform technology with 2nd generation high-k/metal gate transistors optimized for ultra low power, high performance, and high density product applications. In *Electron Devices Meeting (IEDM), 2009 IEEE International*, pages 1 –4, dec. 2009.

[12] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: A power-area simulator for interconnection networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(1):191 –196, jan. 2012.

[13] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 423–428, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[14] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu, and H. Casanova. A case for random shortcut topologies for hpc interconnects. In *Proceedings of the 39th International Symposium on Computer Architecture*, ISCA '12, pages 177–188, Piscataway, NJ, USA, 2012. IEEE Press.

[15] T. Krishna, L.-S. Peh, B. M. Beckmann, and S. K. Reinhardt. Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pages 71–82, New York, NY, USA, 2011. ACM.

[16] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 408–419, Washington, DC, USA, 2005. IEEE Computer Society.

[17] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50 –58, feb 2002.

[18] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33:92–99, November 2005.

[19] A. K. Mishra, N. Vijaykrishnan, and C. R. Das. A case for heterogeneous on-chip interconnects for cmps. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 389–400, New York, NY, USA, 2011. ACM.

[20] N. Muralimanohar and R. Balasubramonian. Cacti 6.0: A tool to model large caches.

[21] U. Nawathe, M. Hassan, K. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *Solid-State Circuits, IEEE Journal of*, 43(1):6 –20, jan. 2008.

[22] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An analysis of on-chip interconnection networks for large-scale chip multiprocessors. *ACM Trans. Archit. Code Optim.*, 7(1):4:1–4:28, May 2010.

[23] C. Seiculescu, S. Volos, N. Khosro Pour, B. Falsafi, and G. De Micheli. CCNoC: On-Chip Interconnects for Cache-Coherent Manycore Server Chips. In *Proceedings of the Workshop on Energy-Efficient Design (WEED 2011)*, 2011.

[24] S. P. E. C. (SPEC). Spec cpu2006, 2006.

[25] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5 –25, jan. 2002.

[26] Tilera. Tilepro64. 2008.

[27] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd annual international symposium on Computer architecture*, ISCA '95, pages 24–36, New York, NY, USA, 1995. ACM.

[28] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.